REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
22-AUG-97

3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE
Complementary Velocity and Heat Transfer Measurements in a Rotating Turbine Cooling Passage

5. FUNDING NUMBERS

6. AUTHOR(S)
Jeffrey Peter Bons

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Massachusetts Institute of Technology

8. PERFORMING ORGANIZATION REPORT NUMBER
97-019D

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DEPARTMENT OF THE AIR FORCE
AFIT/CI

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

14. SUBJECT TERMS

15. NUMBER OF PAGES
201

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT

18. SECURITY CLASSIFICATION OF THIS PAGE

19. SECURITY CLASSIFICATION OF ABSTRACT

20. LIMITATION OF ABSTRACT

Complementary Velocity and Heat Transfer Measurements in a Rotating Turbine Cooling Passage

by

Jeffrey Peter Bons

B.S. & M.S. Aeronautical Engineering
M.I.T., 1988 & 1990

Submitted to the Department of Aeronautics and Astronautics in
Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Aeronautical Engineering

at the

Massachusetts Institute of Technology

September 1997

Signature of Author .... Jeff P Bons ....
Department of Aeronautics and Astronautics
July 25, 1997

Certified by..... Jack L. Kerrebrock ....
Jack L. Kerrebrock
Thesis Supervisor

Certified by.... Edward M. Greitzer ....
Edward M. Greitzer

Certified by.... Gerald R Guenette ....
Gerald R. Guenette

Certified by.... H L McM ....
Hugh L. McManus

Accepted by ....
Mark Drela
Chairman, Departmental Committee on Graduate Students

Complementary Velocity and Heat Transfer Measurements in a Rotating Turbine Cooling Passage

by

Jeffrey Peter Bons

Submitted to the Department of Aeronautics and Astronautics
on July 25, 1997 in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy in
Aeronautical Engineering

ABSTRACT

An experimental investigation was conducted on the internal flowfield of a simulated turbine blade cooling passage. The passage is of a square cross-section and was manufactured from quartz for optical accessibility. Velocity measurements were taken using Particle Image Velocimetry for both heated and non-heated cases. Thin film resistive heaters on the four passage walls allow heat to be added to the coolant flow without obstructing laser access. Under the same conditions, an infrared detector with associated optics collected wall temperature data for use in calculating local Nusselt number. The test section was operated with radial outward flow and at values of Reynolds number, Rotation number, and density ratio typical of applications.

Velocity data for the non-heated case document the evolution of the Coriolis-induced double vortex. The vortex has the effect of increasing the leading side boundary layer thickness while decreasing the trailing side boundary layer thickness. Also, the streamwise component of the Coriolis acceleration creates a thinned side wall boundary layer. These data reveal an unsteady, turbulent flowfield in the cooling passage.

Velocity data for the heated case show a strongly distorted streamwise profile indicative of a buoyancy effect on the leading side. The Coriolis vortex is the mechanism for the accumulation of stagnant flow on the leading side of the passage. Heat transfer data show a maximum factor of two difference in the Nusselt number from trailing side to leading side. An estimate of this heat transfer disparity based on the measured boundary layer edge velocity yields approximately the same factor of two.

A momentum integral model was developed for data interpretation which accounts for Coriolis and buoyancy effects. Calculated streamwise profiles and secondary flows match the experimental data well. The model, the velocity data, and the heat transfer data combine to suggest the presence of separated flow on the leading wall starting at about five passage widths for the conditions studied.

Thesis Supervisor: Jack L. Kerrebrock
Title: Professor of Aeronautical and Astronautical Engineering

Acknowledgements

I would like to thank my advisor, Jack Kerrebrock, for his guidance and support over the last three years. Without his help I would probably still be searching for red herrings at low Reynolds number in an oily test section. In addition to his role as academic mentor, Prof. Kerrebrock has constantly reminded me not to lose sight of my real priorities, as a husband and father. He is a true friend.

The other members of my doctoral committee, especially Dr. Gerald Guenette, have been similarly helpful and supportive. Their enlightening insight and critical comments have given me ample opportunities to stretch my abilities and learn.

I could not have conducted this work without the technical staff at GTL. Bill Ames, James Letendre, and Victor Dubrowski have all been very helpful. Just the same, a graduate student never finishes without considerable assistance from fellow students. A particularly hearty thanks to Takeo Kuraishi for many a thought-provoking discussion. I owe him a considerable debt in CFD consulting fees.

My family has been a great support (and diversion) over the last three years. Nothing takes your mind off thesis work like 4 small children. Cosette, Nicolas, Clarissa, and Zachary, I love you all. A great thanks to my parents also for their support.

Finally, the decision to undertake a PhD program was a joint venture with my dear wife Becky. She has buoyed me up from the day of my first anxieties over qualifiers all the way until the last word written in this document. Her support and love are written on every page. Thankyou Becky, I love you.

Table of Contents

Title..1
Abstract...2
Acknowledgements...3
Table of Contents 4
Nomenclature 5
Chapter 1 - Introduction 6
Figure 1.1 15
Chapter 2 - Experimental Method 16
Figures 2.1 - 2.27 39
Chapter 3 - Analytical Model 66
Figures 3.1 - 3.9 81
Chapter 4 - Results and Discussion 90
Figures 4.1 - 4.27 102
Chapter 5 - Summary and Conclusions.......................................129
References 132
Appendix A 136
Appendix B 155
Appendix C 159

Nomenclature

Bo - Buoyancy param. = $\beta(T-T_\infty)R_m\Omega^2 d/u^2$
d - hydraulic diameter = 4 x area/perimeter
d_p - seed particle diameter
d.r. - density ratio = $(T_w-T_{in})/T_w$
e - local electric field strength
h - convective heat transfer coefficient
H - boundary layer shape factor
J - current density
k - thermal conductivity
l - passage length
M - optical magnification ratio
n - power law profile exponent
Nu - Nusselt number
p - local static pressure
Pr - Prandtl number = ν/α
q - surface heat flux
r or R - radial distance to axis of rotation
R_m - mean radius of test section
Re - passage Reynolds number = du_{in}/ν
Re_x - streamwise Reynolds number = xu/ν
Rot - Rotation number = $\Omega d/u$
Ra_Ω - rotational Rayleigh number
T - local static temperature
u - x component of velocity
v - y component of velocity
v_o - velocity of wall injected boundary fluid
V - complete velocity vector
w - z component of velocity
x - radial (streamwise) direction in passage
y - axis parallel with axis of rotation
z - crossflow direction in passage

α - thermal diffusivity
 - temperature coefficient of resistivity
β - volumetric expansion coefficient
δ - boundary layer thickness
δ_d - boundary layer displacement thickness
ε - surface emissivity
λ - wavelength of radiation
μ - viscosity
ν - kinematic viscosity
θ - boundary layer momentum thickness
ρ - fluid density
 - resistivity
 - seed particle density
σ - Stefan-Boltzmann constant
τ_o - wall shear stress
Φ - electric potential
Ω - rotational frequency

subscripts
BS - back side passage wall
cor - Coriolis
f - fluid surrounding seed particle
FS - front side passage wall
film - fluid conditions averaged from wall and freestream
in - passage inlet conditions
ITO - Indium Tin Oxide thin metal film heater
LS - leading passage wall
p - seed particle
ref - reference conditions
TS - trailing passage wall
∞ - freestream conditions

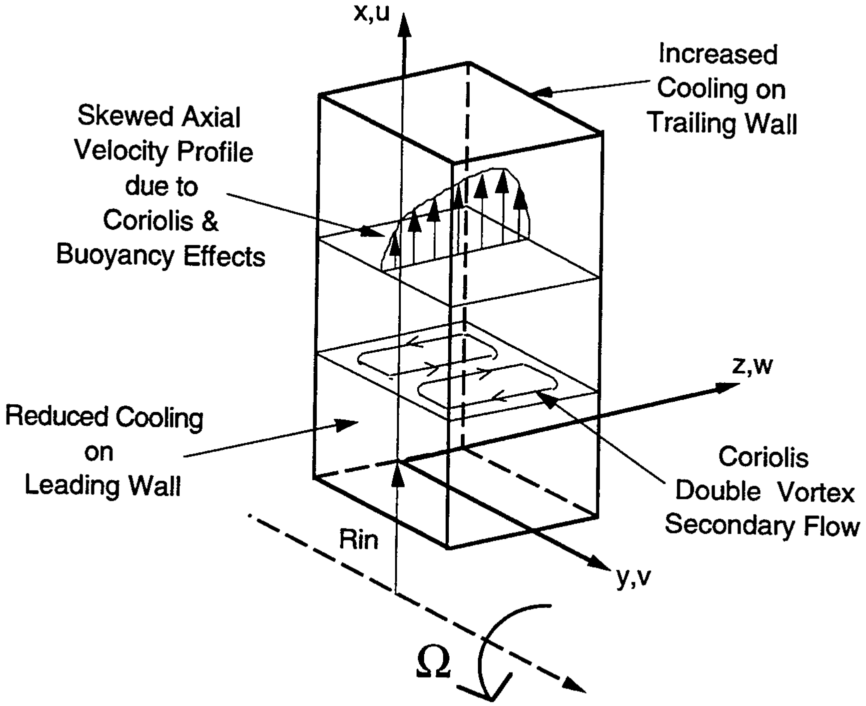
Chapter 1: Introduction

Motivation

The hot section of a modern gas turbine engine is a challenging design environment. The turbine is required to operate with high inlet temperatures and aggressive blade loading to provide high specific power. Aggressive blade loading means high blade speed which converts to high blade root stresses. A high gas temperature means locally higher metal temperatures which in turn create thermal stresses and thermal fatigue problems. These demands conflict directly with the limitations of common (and even exotic) engineering materials. To resolve this conflict, since the 1970s bleed air from upstream compressor stages has been used to cool turbine vanes, blades, casings, and disks. With cooling, engine designers have realized a 300-500°C increase in operating turbine inlet temperatures beyond previous limits.

To cool the rotating turbine blade in particular, cooling air is brought into the rotating disk through intricate mechanical seals and enters the blade through the root. The air then passes through a network of serpentine cooling passages oriented along the span of the blade. Raised steps (or ribs) are commonly used to trip the boundary layers and augment the heat transfer from the metal walls in these internal cooling passages. Even higher levels of cooling are generated on the leading edge of the blade through the use of impingement cooling. Finally, some applications require the use of film cooling to provide low temperature boundary layer fluid on the exterior surface of the turbine blade. In this case, the internal cooling passage plays the additional role of high pressure feed plenum for the film cooling flow. Figure 1.1 depicts all three forms of cooling in a turbine blade schematic. This thesis focuses exclusively on internal cooling.

Modern turbine design depends on reliable cooling schemes to achieve performance levels beyond the capabilities of blading materials alone. Even temporary or partial loss of coolant creates aggravated thermal stresses in the highly stressed rotating parts which can result in catastrophic failure of the entire gas turbine. Because of its critical role, considerable effort has been expended to accurately predict the performance of turbine internal cooling. Design engineers realized early on, however, that because of the rotating turbine frame of reference, standard pipe flow correlations for heat transfer are inadequate. Rotation introduces Coriolis and buoyancy accelerations which create secondary flow vortices and skew the mean flow profile. These flow non-uniformities result in nonhomogeneous heat transfer from the different passage walls depending on their orientation (see schematic of flow in rotating internal cooling passage below).



This complexity forced the first generation of cooling design engineers to rely almost exclusively on empirical correlations.

Objective

With such strong technical motivation, a substantial body of experimental and computational work has been performed to better understand this design problem. Experimental research has succeeded in mapping out many of the qualitative trends in the performance of cooling passages with rotation. But while trends in heat transfer are well understood, the underlying mechanisms causing these rotational effects are as yet unexplored. This is because until recently none of the experimental work has measured the velocity field inside the rotating cooling passage. Only the velocity field data (correlated with heat transfer measurements) provides the complete picture of the rotating cooling passage. Computational work has largely corroborated the conclusions of the experimental work. Flow calculations provide valuable insight, but as yet there is no comprehensive body of heated, rotating velocity data to permit code validation.

Meanwhile, aggressive performance goals have been established for future turbine applications. The next generation of engines must be able to operate at higher turbine inlet temperatures, with increased blade loading. At the same time, combustors are becoming shorter (to reduce engine size and weight), thus combustor exit flows are less uniform with higher turbulence levels. In addition, to avoid viscous mixing losses associated with film cooling flows, designers are relying more and more on the exclusive use of innovative internal cooling designs. All of these initiatives are being pursued, while at the same time goals are set for using less (not more) cooling flow. To achieve these goals, cooling designers must be endowed with the insights gained only by linking the observed heat transfer with the passage flowfield.

The objective of this research is to provide the technical community with the first measurements of velocity and heat transfer in a simulated turbine blade cooling passage. The method used to obtain the velocity measurements was particle image velocimetry (PIV). Thin film heaters on all four walls of the transparent test section provide the necessary constant heat flux wall condition, while infrared imaging of the surface provides the measurement of heat transfer coefficient.

Definition of Flow Scaling Parameters

To simulate the flow environment of a turbine blade's internal cooling passage in the laboratory, the characteristic dimensionless parameters associated with rotating, heated passage flow must be determined. Maintaining these scaling parameters insures the relevance of laboratory findings to the actual operating gas turbine.

The important scaling parameters are derived by non-dimensionalizing the governing equations of motion for non-isothermal rotating flow [16,42]. The averaged Navier-Stokes conservation of momentum equation is shown here for steady turbulent flow.

$$\left(\bar{u}\cdot\bar{\nabla}\right)\bar{u}+\frac{\partial\left(\overline{u_i^{'}u_j^{'}}\right)}{\partial x_j}=\nu\nabla^2\bar{u}-\frac{\nabla p}{\rho}-2\bar{\Omega}\times\bar{u}+\beta(T-T_\infty)(\bar{\Omega}\times\bar{\Omega}\times\bar{R})$$

(The Boussinesq [7] approximation has been applied so that density variations are only accounted for in the buoyancy source term.) This relation can be conveniently expressed in non-dimensional form by using the passage hydraulic diameter, d, as the characteristic length scale and the passage inlet velocity, u_{in} , as the characteristic velocity. The result is shown below for the x-component only. (The asterisk * signifies a dimensionless quantity.)

$$u^*\frac{\partial u^*}{\partial x^*}+v^*\frac{\partial u^*}{\partial y^*}+w^*\frac{\partial u^*}{\partial z^*}+\frac{\partial \overline{u'^{*2}}}{\partial x^*}+\frac{\partial \overline{u'^*v'^*}}{\partial y^*}+\frac{\partial \overline{u'^*w'^*}}{\partial z^*}=$$

$$\frac{\mu}{\rho d u_{in}}\nabla^2 u^*-\frac{1}{\rho u_{in}^2}\frac{\partial p}{\partial x^*}-2\frac{\Omega d}{u_{in}}w^*-\frac{(T-T_\infty)}{T}\left(\frac{\Omega d}{u_{in}}\right)^2\frac{(R_{in}+x)}{d}$$

Several groupings of variables are obvious from this expression. They are defined as the Reynolds number, Re, the Rotation number, Rot, and the Buoyancy parameter, Bo, in this text.

$$Re=\frac{\rho d u_{in}}{\mu} \qquad Rot=\frac{\Omega d}{u_{in}} \qquad Bo=\frac{T-T_{in}}{T}\left(\frac{\Omega d}{u_{in}}\right)^2\frac{R_m}{d}$$

The Reynolds number is the familiar parameter from boundary layer flow analysis. It represents a ratio of inertial to viscous forces in the flow. The Rotation number is peculiar to rotating flows and represents the ratio of Coriolis to inertial forces. The same ratio is sometimes expressed as the inverse Rotation number, or Rossby number. The Buoyancy parameter represents the ratio of centrifugal to inertial forces. It can be expressed as

$$Bo=\frac{\beta(T-T_{in})\Omega^2 R_m d^3}{\nu\alpha}\left(\frac{\alpha}{\nu}\right)\left(\frac{\nu}{ud}\right)^2=\frac{Ra_\Omega}{Pr\,Re^2}$$

where Ra_Ω is a rotational Rayleigh number [42]. Ra_Ω is similar to the Rayleigh number from free convection analysis, the gravitational body force being replaced by a centrifugal body force. Alternatively, the Buoyancy parameter can be expressed in terms of a density ratio and the Rotation number,

$$Bo=d.r.(Rot)^2\frac{R_m}{d} \qquad \text{where} \qquad d.r.=\frac{\rho_{in}-\rho_w}{\rho_{in}}=\frac{T_w-T_{in}}{T_w}$$

The bulk of turbine cooling research reported in the literature uses this density ratio to express the effect of centrifugal buoyancy. For ease of comparison, the density ratio is used here as well. The geometric parameters of relevance are the passage length to hydraulic diameter ratio, l/d, and the mean passage radius to diameter ratio, R_m/d.

Table 1.1 shows values of these dimensionless parameters which are representative of current aircraft and ground-based turbines [15]. The range of values used in this research are also listed for comparison.

Parameters	Current Study	Aircraft	Land-Based
Re	8000 - 10000	25000	up to 800000
Rot	0 - 0.3	0.1 - 0.3	0.1 - 0.4
Bo	0 - 0.95	0.05 - 0.3	up to 8
d.r.	0.0 & 0.27	0.1	0.1 - 0.3
R_m/d	45.8	50	90 - 170
l/d	11.5	12	13 - 25

Table 1.1

The only parameter not well matched is Re. The Reynolds number selected for this study is not a limiting Reynolds number for the facility used. Re up to 124,000 [15] has been demonstrated with this apparatus. The low Reynolds number is due to the relatively large passage diameter (for optical access) and low fluid density (for seeding restrictions), which were both selected for ease of demonstrating the viability of PIV. At Re=10000 the flow is generally considered to be turbulent [16,18], so the character of the flow should not change appreciably at higher Re. Future research can investigate the effect of larger Re now that the fundamental research tool has been successfully demonstrated.

The non-dimensional parameter chosen to represent the heat transfer at the passage wall is the Nusselt number, Nu:

$$\mathrm{Nu} = \frac{h}{k/d} \propto \frac{q_{\mathrm{measured}}/\Delta T}{q_{\mathrm{conduction}}/\Delta T}$$

The Nusselt number is the ratio of the actual heat transfer measured (by convection and conduction) to the heat transfer by conduction alone through a stagnant fluid layer of thickness d. Nu is used exclusively throughout this report to characterize the wall heat transfer.

Previous Work

As far back as the early 1950's (Barua[3] and Boyer[8]), papers were being published on rotating duct flows. Much of the early analyses drew from experience gained studying bend flows and curved passage flows. Hill and Moon [21] showed that the effect of rotation in the orthogonal mode (flow axis perpendicular to rotation axis) was two-fold. First, it induced a double vortex secondary flow and shifted the core flow to the pressure surface for radial outflow. Second, rotation reinforced turbulent fluctuations and enhanced turbulent mixing on the pressure side, while on the suction side rotation retarded turbulent fluctuations. The combination of both effects created higher skin friction and a thinner boundary layer on the pressure surface and the opposite conditions on the suction surface. To isolate the effects of secondary flow and Coriolis-induced instability, Moore [39] varied the width of a rotating rectangular channel while keeping the height constant. The effects due to secondary flow were found to dominate the measured effect of rotation on boundary layer growth and skin friction.

Based on Boyer and others' experimental findings that the rotating passage flow could be adequately characterized by an inviscid core region surrounded by thin wall layers, early analytical approaches employed either a perturbation method (Barua and Moore) or momentum integral analysis. Mori and Nakayama [40] and Ito and Nanbu [25] pioneered the latter method with considerable success.

Interest soon developed to predict the effect of rotation on passage heat transfer. Researchers found that using Reynolds analogy to directly transport the effect of rotation from skin friction to heat transfer was inappropriate. Morris and Ayhan [41] were first to incorporate the effect of centrifugal buoyancy in the interpretation of observed heat transfer phenomena in a rotating circular cross-section cooling tube. They proposed a Nusselt correlation based on a rotational Rayleigh number divided by the square of the Reynolds number (equivalent to Bo). Morris et al persisted with numerous studies over the next decade, documenting the circumferential variations in rotating heat transfer data [43] and the effect of passage cross-section shape [20].

In the late 1980's and early 1990's interest in rotating heated passage flows blossomed with the direct application being turbine blade cooling passages. In 1989, Guidez [16] published the results of experiments using an infrared pyrometer to measure the wall temperature of a rotating passage heated in an evacuated furnace. Much of the work reported before Guidez involved laminar or transitional Reynolds numbers. For this reason, Guidez focused on a range of Re from 15000 to 50000. He found the ratio of trailing side Nu to leading side Nu to be about two for Re=24000, and noted that this ratio decreased with increasing Re. Comparing his results with previous work, Guidez found less than satisfactory agreement. He attributed this to differences in the experimental apparatus: passage inlet and passage shape. This admitted "facility dependence" of experimental data continues to plague the research community today, though to a lesser degree. Guidez also performed a numerical calculation of the flowfield.

Following Guidez were protracted experimental studies by Wagner, Johnson et al at United Technologies Research Center (UTRC) and Han, Zhang, Dutta et al at Texas A&M. The UTRC team methodically studied the effects of Re, Rot, and d.r. [51], flow direction [52], normal ribs [53], skewed ribs [27], stagger angle and rotation direction [28]. While their complete findings are too numerous to mention, their work quickly became the reference point for all subsequent research. Particular results with direct bearing on this thesis are the effects of Rot, Re, and d.r. on radial outward flow. Increasing d.r. augmented Nu on both trailing and leading side. Increasing Re was also found to increase Nu on both sides, contrary to Guidez' finding. This was true even with Nu normalized by a stationary Nu correlation based on $Re^{0.8}$. Finally, increasing Rot raised Nu monotonically on the trailing side, but not so on the leading side. There the Nu first dropped and then increased with increasing Rot. Wagner et al [51] were the first to use the density ratio to characterize the effects of centrifugal buoyancy.

Meanwhile the Texas A&M team explored the effects of passage shape [12] and thermal boundary conditions [17,18] . The UTRC facility is an exclusively constant wall temperature apparatus, while the Texas A&M facility can be operated in either constant wall temperature or constant wall heat flux mode. There is yet some disagreement over which boundary condition is more representative of the turbine cooling passage. Han et al [18] found that the constant heat flux mode had higher Nu on both leading and trailing side than the constant wall temperature case for the same density ratio. They reasoned that because a constant heat flux gives uneven wall temperatures, the buoyancy forces are unbalanced and create better cooling everywhere as they destabilize the passage flow. Han and Zhang [17] also experimented with uneven wall heating with a similarly marked effect. Their Nu data is normalized by the same stationary Nu correlation ($Re^{0.8}$), which they found to adequately remove Re dependency in the rotating data.

In addition to the work at UTRC and Texas A&M, other experimental studies with varying levels of sophistication have been reported. Of specific interest is recent work carried out at MIT on a facility originally designed for impingement cooling research [34]. In 1994 and 1995 three experimental programs were conducted [2,29,15] exploring effects of aspect ratio, skewed ribs, uneven wall heating, and flow direction. The results served as a point of departure for the work in this thesis.

On the computational front, researchers had also been active applying Navier-Stokes solvers with k-ε and Reynolds stress turbulence models to rotating flows. Like the experimentalists, initial computational studies did not treat the effect of buoyancy [22,31]. They did, however, predict much of the experimentally measured effects of Coriolis. One additional insight was the development of an additional vortex pair (making 4 vortices in all) at high Rot [24,31]. Guidez [16] and Prakash and Zerkle [45] finally added the full effects of buoyancy with surprising results. Prakash and Zerkle predicted flow reversal at high Rot on the leading wall. Since then, many other studies have been conducted exploring the effects of ribs [6], passage shape [35], flow direction, and thermal boundary condition [32] among others.

All of the heated rotating computations have used the flow prediction to estimate Nu, which is then compared with experiment to assess code validity. A lack of detailed flow measurements in the heated, rotating frame has prevented direct comparison with velocity data. This is not to suggest that velocity measurements have not been attempted. As far back as Moore [39] and Wagner and Velkoff [54], Coriolis secondary flows and boundary layer growth have been measured using existing hot wire technology. The advent of non-intrusive laser flow measurement techniques greatly facilitated this effort in the last decade. In 1991 Uellner and Roesner [50] and Berg et al [4] both published velocity measurements from rotating (non-heated) facilities. Uellner and Roesner used laser speckle velocimetry (LSV) and Berg et al applied the laser two-focus method. Their results corroborated earlier hot-wire measurements. In 1995 and 1997, Tse et al [48,49] used a test section configuration similar to the UTRC apparatus to obtain detailed velocity contours with 3-component laser doppler velocimetry (LDV). Again, their test section is not heated. Most recently, Hsieh et al [23] obtained velocity data using LDV in a rotating passage with heating on the leading and trailing walls only. Measurements are presented only along the passage centerline and show a limited effect of rotation up to the maximum Rot of 0.06. While their success at tackling a difficult problem is commendable, their test section bears little resemblance to a turbine cooling passage. The 25d long square passage has an inlet radius of 1d, with heat added at a radius of 6d (turbine cooling passages typically have an inlet radius >50d). Also, the 'U' shaped passage is not oriented radially from the axis of rotation, creating multiple components of Coriolis and buoyancy. In addition, the velocity data indicate a far greater effect of the trips in the passage than the rotation effect at such low Rot.

Contributions of this Thesis

The present study contains the first reported global velocity measurements in a heated, rotating test section. The results document the Coriolis vortex and the effect of buoyancy. In addition, unlike hot wires or LDV, the PIV velocity measurement technique provides a two dimensional plane of the flowfield with each image. So rather than relying solely on statistical samples, PIV allows insight into the unsteady character of the flow. The velocity data alone would be a significant contribution to code validation efforts and improved physical understanding, but this is not all. High resolution heat transfer measurements are also presented with the same test section operating at the same conditions. This allows a direct correlation of flow phenomena to heat transfer phenomena without the ambiguities of different experimental facilities. Finally, the test section geometry is representative of a turbine blade cooling passage and it is operated well within the industrial parameter space. So, the findings have direct relevance to turbine cooling design. These are the specific contributions of this thesis.

Thesis Outline

Because a major objective of this research effort was the demonstration of a new capability (PIV in a rotating, heated test section), the passage design was simplified. The cross-section is square and there are no heat transfer augmentation ribs. The model was tested in only one flow direction (radial outward) and the stagger angle is 0° throughout. The thermal boundary condition is nominally constant heat flux and the inlet condition is a tapered orifice with only 1.25 diameters of unheated starting length.

The experimental apparatus is described in detail in the next chapter. The PIV image acquisition and processing is also outlined in chapter 2, along with the wall temperature measurement and Nusselt calculation procedure. Chapter 3 contains the outline of an analytical flow model that was used for data interpretation. Chapter 4 presents all of the experimental data with discussion and model comparisons. Finally, the entire work is summarized with conclusions in chapter 5. The appendices contain copies of the various computer programs used for data analysis.

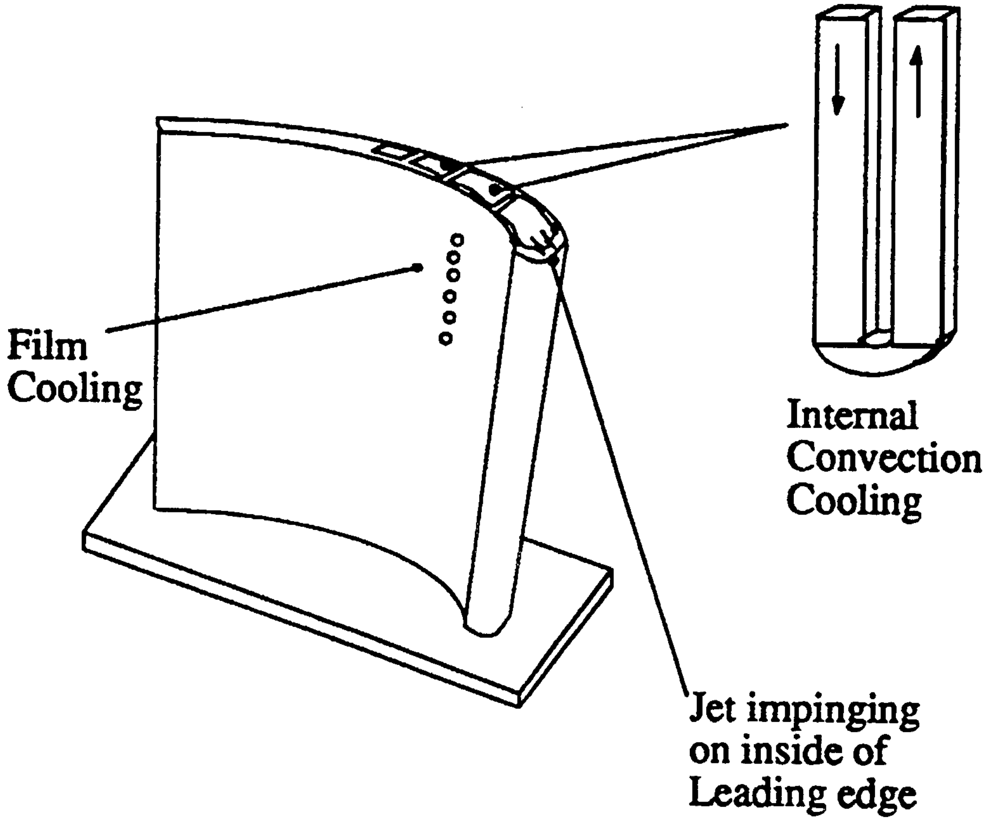

Film Cooling
Internal Convection Cooling
Jet impinging on inside of Leading edge

Figure 1.1: Schematic of Turbine Blade Indicating
3 Forms of Cooling: Impingement, Internal, and Film [15]

Chapter 2: Experimental Method

Introduction

The experimental facility used for this study was originally designed by Kreatsoulas[34] for impingement cooling studies. It was subsequently modified by Barry[2], Jones[29], and Govatzidakis[15] for parametric internal cooling research. The facility can be operated over a range of Re, Rot, and d.r. which match actual gas turbine operating conditions. Other variables include: flow direction, rotation direction, passage geometry, and stagger angle. The testing done for this study used radial outward flow, and forward rotation only at 0° stagger angle. Figure 2.1 shows a schematic of the facility. The test section is mounted at the end of a 0.4m radius arm which spins in an evacuated chamber. Cooling fluid enters and exits the rotating arm and passage through an arrangement of seals located on the shaft. Heat is applied to the passage via resistive heating and the passage surface temperatures are measured with an infrared scanner.

This chapter contains a detailed description of the experimental facility as well as modifications that were made to provide optical accessibility for PIV measurements. Finally the PIV image processing and Nusselt calculation procedures are outlined in detail.

Facility Overview

The rotating arm is mounted on a shaft which occupies the cavity of a large vacuum tank. The arm is constructed from two rigid parallel steel plates and has a diameter of 0.81m. The shaft and rotor are driven by a 7.5hp, 0-60rps variable speed motor that is controlled by the data acquisition system to within ±0.03rps. A balance ring mounted on the rotating arm permits fine control of vibration levels associated with rotation. The rotation speed controller and vibration suppression system were critical to obtaining accurate and reliable optical images for use in PIV. The shaft speed is measured using an optical encoder whose signal is processed by the data acquisition system. The once per revolution signal from the encoder is used to trigger the laser/camera synchronization for PIV and the IR data acquisition for wall temperature measurements. The entire rotating arm is housed in a cylindrical vacuum chamber. The vacuum is created by a Stokes 145H-11 Microvac vacuum pump and typical pressures during testing are approximately 4Torr.

Operating in a vacuum avoids windage losses during rotation and more importantly eliminates passage heat transfer loss due to external convection.

The cooling fluid circuit can be operated in either open or closed loop. In open loop mode compressed air is fed to the rotating passage through a series of valves and a regulator. The actual mass flow is measured using a Micromotions CMF025-319 massflow sensor. The operating pressure just upstream of the test section is measured with an Omega PX213-100G5V pressure transducer. Since the pressure transducer is not located on board the rotating arm, the pressure drop through the shaft seals was measured and is deducted from the transducer measurement to give an accurate reference pressure at the test section inlet. Typical pressure drop was 0.6psi at 60psia operating pressure. Both the massflow and pressure analog outputs are sampled continuously by the data acquisition system. With these two measurements, and a fluid reference temperature, the passage Re and Rot are calculated real time throughout testing. After exiting the rotating frame through the shaft seals, the air is exhausted to atmosphere. The closed loop mode is used when operation with a gas other than air is deemed necessary to achieve elevated Reynolds number. For example, Freon-12 can be used in the closed loop mode to give a factor of nearly four in Re at constant Rot. The open loop mode was employed exclusively for this testing primarily because air is compatible with the seeding selected for PIV.

Four type-K thermocouples are located on the rotating arm and give redundant measurements of passage inlet and exit temperature. The signals are brought to the stationary frame via a standard slip ring assembly. To avoid signal corruption associated with copper-to-thermocouple wire junctions at the unknown temperature of the slip ring assembly, on-board junction compensation was employed using Omega Temperature Transmitters, model TX-91K. Essentially, these devices produce a current proportional to the thermocouple signal and thus avoid noise associated with spurious voltage drops. The current signals are brought through the slip ring to a series of precision resistors and the voltage drop there is sampled by the data acquisition system. The transmitters are calibrated to within ±1° C using a two reference temperature calibration procedure. The pressure and massflow meter are calibrated using in-line analog meters for pressure and massflow (a rotameter) respectively to within ±1%.

Power slip rings provide the current necessary for resistance heating of the test section. A filtered DC power supply provided a stable operating current up to 2Amps with ±0.1% ripple. The current was monitored with a 0.7Ω shunt resistor and a digital voltmeter. A current drift during testing of ±2.5% was the largest source of error in the current measurement.

The data acquisition system consists of two National Instruments Data Acquisition boards operated using the Labview software. The two boards provide a total of 12 differential analog inputs with 12bit/channel resolution. The output channels on the boards are also used to control the rotational frequency of the shaft, provide synchronization TTL signals to the camera and laser, and operate the IR traverse system.

The Test Section

It was clear from the beginning of this research that creating a test section that allows both optical access and an applied heat load would be a significant technological undertaking. The quartz test section with thin metal coating used in the study was only selected after a thorough review of suitable candidates. The quartz test section is 115mm long and is manufactured from drawn square-bore tubing. The ID is 10mm x 10mm and the walls are ground and polished to a 1.5mm thickness. Figure 2.2 is a schematic of the finished product. The center 90mm along each of the four sides is coated with a 10mm wide swath of Indium Tin Oxide (ITO). The coating thickness is nominally 4500Angstroms ±5% and it is applied with electron beam sputtering to the outside of the test section as shown. To prevent film oxidation (and the associated resistance rise) at elevated temperatures, a second thinner coating of SiO_2 is applied on top of the ITO through a proprietary index-matching process by the manufacturer, Thin Film Devices Inc. With this coating, the film can operate up to 200°C without noticeable degradation. Heat fluxes up to 5 W/cm^2 were achieved during preliminary testing. Above this level voids appear in the film due to ITO vaporization and the resistance quickly rises to infinite. The operating level for these tests was approximately $1W/cm^2$. 8mm long Cr/Ni/Au busbars at either end of the ITO film provide uniform electrical contact to the ITO. Contact to the busbar is made using a star-shaped wave spring which is clamped securely at both ends (see Fig 2.2) of the passage. The four sides are thus connected in a parallel circuit. Beyond the busbar is 4.5mm of non-coated quartz. This free end is potted using room temperature curing epoxy into a specially designed fixture which mounts to the end of the rotating arm in the vacuum chamber. The epoxy provides the needed seal and absorbs any mechanical misalignment or shifting during assembly and rotation without overstressing the brittle quartz test section.

Due to a manufacturing complexity, the front side wall ITO sheet thickness is approximately 20% greater than the ITO thickness on the leading and trailing walls (which are roughly equivalent). This creates a lower resistance here, and a higher heat flux since more current passes through this part of the parallel resistance circuit. The back side wall's ITO thickness is conversely 20% less than the leading and trailing walls, causing a corresponding decrease in the heat flux here. The result is a higher heat load on the front internal wall by nearly 30% after a 3D conduction algorithm (later this chapter) accounts for temperature smearing through the test section walls. The effect of this heating imbalance is discussed in chapter 4.

During the design phase considerable energy was expended to determine the minimum acceptable wall thickness given the predicted stresses in the test section. Stresses due to differential pressure, thermal gradients, and rotation were all factored into the design. The 1.5mm wall thickness was able to sustain pressures up to 60psi and none of the test sections were lost during testing. Careful handling of the test section before and after installation is critical. Figure 2.3 is a schematic of the installed test section. Also indicated is the upstream flow conditioning screen, tapered inlet, and inlet temperature measurement port.

ITO Film Properties

The ITO film and quartz substrate provide 80% transmission at the visible wavelength of the laser used. Since the quartz is opaque in the IR spectrum, the capability of the IR scanning temperature measurement system was not compromised with the use of a transparent test section. The emissivity of the ITO was measured using an IR scanner with the coated test section filled by a reference isothermal copper block. Its emissivity is approximately 0.27. The emissivity of the quartz is nearly 0.9.

The ITO resistivity has a non-negligible dependency on temperature. The resistance of a sample was monitored in an oven over a 200° C range and the temperature coefficient of resistivity was calculated to be α=310 e-6/°C, where

$$\rho(T) = \rho(T_{ref})[1 + \alpha(T - T_{ref})]$$

A variable resistance will produce a non-uniform heat flux distribution on the passage exterior wall if the temperature is not spatially uniform. A local hot spot will have a higher resistance (α >0) and the current will tend to flow around it, creating a lower local heat flux. Since we expected temperature gradients up to 50°C around the test section circumference, the effect of this variable resistivity needed to be assessed. To do this a finite difference scheme was created to solve the conservation law for the current density, J, in the ITO film on each side of the test section.

$$\nabla \cdot \vec{J} = 0 \qquad \text{where } \vec{J} = \frac{\vec{e}}{\rho} \qquad \text{and} \qquad \vec{e} = -\nabla \Phi$$

Since the ITO film has a thickness of 4500Angstroms << 90mm length and 10mm width, the above relation is solved in only two dimensions:

$$\frac{\partial}{\partial x}\left[\frac{1}{\rho}\frac{\partial \Phi}{\partial x}\right] + \frac{\partial}{\partial y}\left[\frac{1}{\rho}\frac{\partial \Phi}{\partial y}\right] = 0$$

The resistivity is a function of temperature which is in turn a function of x and y, so

$$\frac{1}{\rho}\frac{\partial^2 \Phi}{\partial x^2} + \frac{1}{\rho}\frac{\partial^2 \Phi}{\partial y^2} - \frac{1}{\rho^2}\frac{\partial \Phi}{\partial x}\frac{\partial \rho}{\partial x} - \frac{1}{\rho^2}\frac{\partial \Phi}{\partial y}\frac{\partial \rho}{\partial y} = 0$$

T(x,y) on the ITO surface is known from the IR measurements and the relation for ρ(T) has already been measured empirically. So, the above differential equation is discretized with appropriate boundary conditions at the four ITO edges and solved using an underrelaxed Jacobi iteration scheme. Once the Φ distribution is determined, the local heat flux is,

$$q = \frac{|\vec{e}|^2}{\rho} = \frac{|\nabla \Phi|^2}{\rho} = \frac{1}{\rho}\left\{\left(\frac{\partial \Phi}{\partial x}\right)^2 + \left(\frac{\partial \Phi}{\partial y}\right)^2\right\}$$

A copy of the code is at Appendix B. The ρ(T) variation has an effect of <1% on the local heat flux distribution for the testing condition reported here, but it is accurately accounted for in the Nusselt calculation.

The PIV Measurement System

The laser is a QuantaRay PIV series Nd:YAG manufactured by Spectra-Physics. It emits two sequential 200mJ pulses with a nominal pulse width of 10ns. The radiated energy is frequency doubled to obtain a green beam at 532nm. The pulse separation can be varied from 100ns to 100ms. Pulse separations of 25-35μs were used in this research. With a typical flow velocity of 3 m/s in the test section, the 10ns pulse essentially freezes the motion of seed particles while the pulse separation of 30μs gives nearly 100μm of motion between pulses. The pulse separation was verified to a minimum accuracy of 100ns. The exit beam diameter is approximately 1cm.

The focusing optics are mounted to the front face of the vacuum tank to avoid vibration-induced misalignment. The beam reaches the optics through a series of three flat mirrors with a combined throw distance of approximately 1m. Figure 2.4 contains a schematic of the optical path. To produce the light sheet, a 25mm diameter spherical lens with focal length=750mm is positioned outside the vacuum tank with its focal point precisely set to the center of the test section passage when it is in the camera's field of view. A mirror located 5cm after the spherical lens directs the narrowing beam in the required angular orientation to bisect the test section. Approximately 16cm beyond this mirror is the 50mm x 50mm square cylindrical lens. This final lens has a negative focal length of -100mm and thus expands the beam in only one direction to produce a planar sheet normal to the axis of rotation. Figure 2.5 shows the coordinate axes fixed to the rotating test section along with the components of velocity. These axis and velocity nomenclature are used in all subsequent discussions. From this figure, the laser sheet illuminates the x-z plane of the test section.

The sheet height when it intersects the test section is about 4cm. The height can be adjusted by moving the cylindrical lens relative to the spherical lens on an optical track. The position of the sheet relative to the passage x/d can also be varied using the two point micrometer adjustments on the final mirror in the optical path. Both the spherical lens and the final mirror are mounted on an optical traverse with micrometer adjustment so the sheet can be moved from side to side in the test section passage ($0<y/d<1$). Two 1.25cm thick glass windows cover the entrance port for the laser beam and the camera view port (Figure 2.4).

Since the PIV system was employed to measure secondary flow features with finite spatial extent, an accurate measure of the sheet thickness was desirable. To obtain this measurement a flat (2-D) nozzle was fashioned from 1.25cm diameter copper tubing. The nozzle exit width is approximately 0.8mm. This nozzle was installed in place of the test section and in full view of the camera. Seeding was then added to a constant flow of air from the nozzle and the laser sheet (at a nominal power setting) was traversed across the fixed flat jet. The light sheet thickness was then deduced from the range of traverse over which particles were visible in the camera. Using this method, the sheet thickness was estimated at 0.4 ±0.1mm. Because the laser beam has a 60% gaussian profile, the beam intensity actually varies through the sheet thickness and some faint particles were visible over a larger range of traverse motion (suggesting an ultimate sheet thickness of nearly 1mm). However, the bulk of visible particles were in the center 0.4mm of the laser sheet. This finite sheet width prevents velocity measurement closer than 0.5mm to the passage side walls (y/d=0 & y/d=1) as laser reflections obscure the field of view.

Flow Seeding

Since the cooling fluid (air) is relatively free of particulate, seed particles are added to the flow just outside the vacuum tank. Gonesh #2 incense is used as the seed. Sample particles were sized with a Particle Measurement Systems particle sizer (using Mie scattering) at 0.2 to 0.5μm [9]. Gonesh had been used previously in the laboratory for transonic flow visualization with considerable success. The small particle diameter was desired based on a calculation of the particle slip velocity in the rotating centrifugal field. If the particle density is much larger than the fluid density (and Stokes flow about the particle is assumed), a momentum balance for the seed particle in the flow gives [37]:

$$\rho_p \frac{\pi}{6} d_p^3 \frac{dv_p}{dt} = 3\pi\mu d_p \left(v_f - v_p\right) + \rho_p \frac{\pi}{6} d_p^3 \Omega^2 R$$

$\Omega^2 R$ is the centrifugal body force on the seed particle. The critical parameter is v_f-v_p , the difference between the fluid velocity and the seed particle velocity, which must be minimized for accurate flow measurement. Solving this relation for v_f-v_p as a function of time produces an expression with an asymptotic value of

$$\left|v_f - v_p\right| = \frac{d_p^2 \rho_p}{18\mu} \Omega^2 R$$

Using this expression, a particle diameter of 1μm was selected to provide a $|v_f - v_p|/v_f < 1\%$ at the maximum possible rotation frequency of 30rps. Gonesh incense is more than adequate for this application, especially since the rotational frequency used in practice was one third the maximum allowable value.

Camera Parameters

The camera used to obtain the PIV images is a PULNIX TM-745 CCD camera. The pixel resolution is 480 (V) x 752 (H) and the S/N ratio is 50dB min. A Sigma macro-lens (90mm focal length) mounted to the camera housing is set to a nominal F# of 12 for the PIV images. This gives a magnification (image/object size) of M=0.81. The horizontal lines of the output frame from the camera are interlaced. To assure a properly interlaced image, the sampling rate must be an integer multiple of 2 times the 1/60 sec framing rate. For this reason, a rotational frequency of 10 rps was selected for the rotating arm (the arm and camera are synchronized for image acquisition). To obtain consistently interlaced images it is critical that this rotational frequency be maintained throughout the testing period. Thus the need for the <0.2% control of the driving motor rotation. The camera images are acquired using an ITEX Image Capture board and associated software. The images are stored as TIFF-format files (8 bit resolution) for subsequent processing.

Significant levels of stray light are reflected from the walls of the test section during operation of the laser. This optical noise can render the entire field of view useless without appropriate precautions. Accordingly, a beam block with 10mm wide gap was installed on the laser entry side of the test section and an adjustable-gap block was installed on the camera view side (Figure 2.6). An additional black background was also situated behind the test section to eliminate stray reflections there. To accurately identify the region of the passage which is actually in the field of view, a grid was installed inside the test section and the adjustable blocks were set to a gap width of 8mm ($-0.4 < z/d < 0.4$). The absolute position of this 8mm wide gap relative to the test section was measured to an accuracy of ±0.1mm (the z position) using indicial marks on this same grid. The sheet position in the y direction was determined by moving the sheet back and forth inside the test section and noting when the sheet touched the side walls. The midpoint of these two positions indicates $y/d=0.5$ (the center of the passage). The accuracy of this method is estimated at less than half the sheet thickness, or <0.2mm. Even when finely balanced there is some lateral vibration of the test section. Using a fine wire insert into the passage, vibration induced lateral motion was measured to be <±0.1mm when well balanced.

The camera was positioned to optimize the competing requirements of field-of-view and spatial resolution. The result is a maximum available window of 8.2mm in the test section axial (x) direction and 10.7mm across the test section width (z direction). While this theoretically allows access to the entire 10mm test section width, there are several practical limitations to this available field of view. First, though pains were taken to shield the camera from stray light scattered from the edge walls of the test section, there is significant optical noise near the edges which increases significantly as the light sheet approaches the side walls (y ≈ 0 & d). This limits the available view for noise-free particle correlation by 2-4mm on the width, depending on the y location of the laser sheet. This problem is greatly exacerbated when the test section is rotating, since both images of the test section must now be contained in the same 10.7mm wide camera frame. The edge glare from one image overlaps otherwise usable area from its double-image and vice versa. This occurs along both sides, in effect doubling the obscured area. As a result, the processed PIV images generally show vectors over 3-7mm of the 10mm test section width (z direction).

The camera is mounted on a fine pitch screw traverse which allows for camera positioning along the axial (x) direction of the passage. The traverse is accurate to ±0.03mm. The camera does not have the flexibility to traverse back and forth in the y direction to maintain focus as the laser sheet is moved from the front side wall (y=d) to the back side wall (y=0) of the passage. To maintain focus, the lens was adjusted for each new position of the sheet. The magnification adjustment was measured using a square grid imaged at both extremes of the range. The variation in magnification is ±4% (0.8%/mm) and is accounted for in the image processing software.

When the passage is rotating, the laser and camera must be synchronized properly to yield a useful image. To do this, first the trigger signal from the optical encoder is read by the data acquisition system. A delayed pulse (based on the rotational frequency) is then sent to initiate the camera frame acquisition. 33.8ms after this pulse, another pulse initiates the lamp on the first laser cavity which is then Q-switched (180μs later) to give the first laser burst. This elaborate timing sequence insures that the test section is located in the camera field of view when the laser fires. The second laser burst is triggered similarly with a time separation equal to the desired laser burst separation. All of these timing signals are handled by the Stanford Research Systems Inc. Four Channel Digital Delay/Pulse Generator which generates up to four TTL pulses with an accuracy of 5 picoseconds. Since the rotational frequency is controlled to within ±0.03rps, the triggering sequence continues uninterrupted throughout the many hours of testing required to obtain the PIV images. For image processing, it is critical to have an accurate measurement of the rotational frequency for each image. Rather than acquire a simultaneous measurement of Ω, it is estimated based on the test section position in the camera frame. Since the trigger sequence is initiated only once at the beginning of a testing cycle, as Ω varies the test section moves predictably out of the camera field of view. Measuring this shift in pixels on the actual image gives a much improved uncertainty of <±0.001rps.

Beam Alignment

Also of critical importance is the laser sheet alignment relative to the plane of rotation of the passage. Even a slight angular misalignment between the two planes results in a second image which is at a different y/d position than the first, with a different set of seed particles. This greatly hinders the particle pairing or correlation for velocity determination. The solution to this potential problem is the alignment tool shown in Figure 2.7. This tool consists of two fine 13μm diameter wires which form a cross between two long 3mm diameter posts. With this alignment tool inserted into the passage (and the posts situated at the corners of the passage) the sheet illuminates the cross during rotation within full view of the CCD camera. As such, misalignment between the sheet and the plane of rotation is readily apparent. Adjustments in the sheet orientation are made by moving the final mirror. A typical double cross image after alignment is shown in Fig 2.8, where the majority of the sheet strikes the same portion of the cross in both images.

PIV Image Processing

Particle Image Velocimetry (PIV) is an optical measurement technique that relies on sequential bursts of light to provide two or more frozen images of particles in a flowfield. Particles are paired with their double (or correlations are performed) to yield the net motion in the two dimensions of the light sheet. An accurate knowledge of the pulse time separation then provides two components of velocity. The principle is straightforward, but the application is not always simple, especially when high accuracy and spatial resolution are desired. This section outlines the application of PIV employed in this research effort beginning with essential background on PIV in general.

In his 1991 review paper, Adrian [1] summarized many of the important features necessary for accurate velocity measurements using PIV. Since these criteria were used to optimize the PIV images in this work, they are summarized here for completeness. For a diffraction limited lens, the point response is an Airy function with a diameter,

$$d_s = 2.44(1+M)f\#\lambda$$

In this case, M=0.8, λ=532nm, and F#=12 combine to give d_s = 30μm (just 2 pixels in image space). The image of a finite diameter particle is the convolution of the Airy function with the particle geometric image, Md_p . Approximating both functions by Gaussians leads to the following approximate function for the image diameter.

$$d_e = \sqrt{M^2 d_p^2 + d_s^2}$$

With d_p = 0.3μm for the Gonesh seed particles, $d_e \cong d_s$ in this case. The lens depth of field is determined by the relation,

$$dz = 4\left[1 + \frac{1}{M}\right]^2 f\#^2 \lambda$$

dz≅1.2mm for this application, which is greater than the measured light sheet thickness of 0.4mm. Accordingly, there is no noticeable blurring in the image. If the field of view is too large, the paraxial optics assumptions are no longer valid. To insure that this does not happen, we desire the ratio of maximum field dimension to the object/lens distance to be <0.1. This application has a ratio of 0.5cm/15cm = 0.03 < 0.1.

The source density, N_s , of a PIV image is defined as the mean number of particles in a cylindrical volume created by the intersection of the light sheet and the circle of image resolution, d_e/M.

$$N_s = \rho_{part} t_{sheet} \frac{\pi d_e^2}{4M^2}$$

If $N_s \geq 1$ the particle density is too high to distinguish individual particles and determine velocities from a correlation or tracking scheme. Instead the flow is better suited for Laser Speckle Velocimetry which analyzes the interference patterns of multiple particles to determine velocity [50]. $N_s \approx 0.02$ for this application, so PIV is appropriate.

The image density, N_I , is defined as the particle density in a typical correlation interrogation box (to be explained later).

$$N_I = \rho_{part} t_{sheet} \frac{\pi d_I^2}{4M^2}$$

If N_I <<1 the particles and their double images are sufficiently isolated from other particles to allow an unambiguous pairing of individual particles to determine motion and thus velocity. This is known as Particle Tracking or Low Density PIV and was used with some success in the early stages of this research program. If N_I >>1 the particles are too close in proximity for individual particle tracking. Instead, a correlation of particle patterns in the flow is performed to yield one vector per correlation box. Each correlation box may contain 10-20 particles, but their individual velocities are consolidated into a single mean vector. This second method is called High Density PIV. With time, the high density method completely replaced the low density method as the procedure of choice for reasons that will become apparent later. Because the application of low density PIV provided the groundwork for the later use of the high density method, it will be described first followed by the high density method.

Particle Derotation

Before describing either of these PIV methods, a procedure will be outlined which is unique to this rotating PIV application. This procedure, used in both the high and low density methods, involves removing the rotating motion from the double image. Since this is the first reported application of PIV to a rotating test section, this procedure deserves a detailed treatment. In stationary PIV, at times a "bias velocity" is added to the second image to eliminate directional ambiguity (in vortical flows for example). One method of doing this is to displace the second image a known amount using an oscillating mirror in the imaging optical path (Jefferies[26]). This creates an offset, or bias, between the two particle fields which insures a unidirectional velocity field (the bias velocity is typically selected to be larger than the maximum expected reverse flow velocity). Once the vectors are determined, the known bias distance can be removed leaving only the actual flow velocity.

Because the camera and laser are stationary in this experiment, the test section rotation provides an inherent "bias velocity" in the PIV images. The second image is displaced by an angle $\Omega\Delta t$ about the axis of rotation. To remove this angular displacement, Ω, Δt, and r (the radius from the center of rotation to each particle) must be determined. Then the velocity vector field can be adjusted by this amount to leave only the desired particle motion relative to the passage. Ω and Δt are precisely measured quantities for each image, but the position, r, is unknown unless the center of rotation is measured in the camera frame of reference (pixel space).

To determine the center of rotation in "pixel-space", the alignment tool (Fig. 2.7) is again inserted into the test section. The radial position of the wire cross on the inserted alignment tool was measured to an accuracy of 0.1mm. Then, while rotating, a double image is taken of the cross illuminated by the laser sheet (Fig. 2.8). The two cross images are then correlated to determine their relative distance. Using a cross for this procedure produces a far superior correlation accuracy than, for example, a single illuminated wire. The correlation peak for a cross is very well defined permitting an accurate peak location of <0.3pixel (Fig 2.8 shows the correlation map for a cross image). With the cross radius, angle of motion ($\Omega\Delta t$), and net cross motion known, the calculation of the center of rotation is simple geometry. This calculation is performed by the processing code in the subroutine crclcntr.m (Appendix C). With the center of rotation identified in pixel space, the particles in subsequent PIV images can be "derotated" an angle $\Omega\Delta t$ about this pixel center of rotation provided the camera is not moved from its alignment position. When the camera is moved to another x/d using the vertical traverse, the processing software accounts for this motion and shifts the pixel center of rotation accordingly.

Individual uncertainties in Ω, Δt, the cross radius, and the inter-cross distance measurements combine to give an aggregate uncertainty of <0.2% on the pixel center position. When this center is then used to derotate subsequent PIV images, a corresponding "bias" error of ±4% in velocity is introduced. Notably, this is not an error in the traditional sense, since it is applied equally to all images processed with this calculated pixel center. For example, if the pixel center determination uncertainty results in a +3.2% error in velocity vectors, all images processed with this pixel center are subject to the exact same +3.2% error. Thus it is rather a "bias" for all images in the same test run.

Particle Tracking (Low Density) PIV Method

To exercise the low density method, the acquired TIFF-format PIV image is converted to a matrix of intensity values. Figure 2.9 shows a typical double exposure image. All of the codes related to PIV image processing were written in the Matlab programming environment (Appendix C). The images are then processed as follows. First, the image's pixel intensity map is filtered to identify pixels with an intensity above some predetermined threshold value. These high intensity pixels are associated with light scattered from seed particles in the flow. Second, "particles" (high intensity pixels) from the first exposure (first laser pulse) are "derotated" an angle $\Omega\Delta t$ into the second image using the derotation scheme outlined earlier. The uncertainties in Ω and Δt for each individual image combine to give <1% additional uncertainty to all vectors in the image. Again, since this derotation is applied to all particles in an image equally, the actual uncertainty arises when making comparisons between images, not between vectors in a given image.

To eliminate ambiguity between first and second image particles (and thus know which ones to derotate), this step assumes that the two images are not overlapped. This places an onerous constraint on the experimental parameter space since non-overlap only occurs for high rotation and low fluid velocity (Reynolds number), or if the viewable portion of the test section is reduced to a small fraction of the available width (from 8mm to 1 or 2mm).

Once the particles have been derotated they are paired with likely matches based on an expected range of angle and velocity magnitude. Finally, the intensity signatures of the paired particles are correlated (each with its double) to yield a Δx, Δz estimate of motion. To find the tip of the central peak in the correlation the intensity centroid of the top 25% of the peak is calculated. The uncertainty of determining this peak is estimated at <0.5pixels. For 8 pixels of net motion, this represents a ±6% uncertainty in each individual vector. Other methods of determining the correlation peak were attempted, including a sophisticated spatial integral of a double gaussian, but resulted in comparable accuracy due to the limited pixel resolution and the sometimes irregular shape of the correlation peaks. Obviously, with greater pixel resolution (or with larger particle separation) this uncertainty can be reduced significantly. Figure 2.10 shows the result of processing the image in Figure 2.9. This image was taken near the test section side wall (y/d= 0.09) and shows a strong mean velocity in the direction of rotation. This is evidence of the trademark Coriolis vortex which will be dissected in detail in chapters 3 and 4.

Though this method is fairly robust, it also has serious limitations. As mentioned above, the non-overlap requirement stipulates a high Rot and low Re and essentially precludes operation in the Re range of interest (>10000). In addition, the particle identification step becomes tedious when optical background noise is present. Often, manual deletion of erroneous particles is required to "clean up" an image. Finally, since the pairing is done one particle at a time, the process is time consuming. A typical noisy image may take 10-20 minutes to reduce to a velocity field. To improve on this figure of merit a number of standard techniques were employed such as "sliding and stretching" the intensity pattern before correlation and consulting the velocity of neighboring particles when trying to resolve multiple matchup possibilities. Even still, the limitations on Re and Rot ultimately drove the switch to the high density correlation technique.

High Density PIV Method

In high density PIV the image is divided into interrogation boxes (typically 75 x 75 pixels), each box containing 15-25 particles. Figure 2.11 shows a sample high density image. To process the image, the entire box pattern is "derotated" as before and then cross-correlated with the section of image it overlaps once derotated. Fig 2.12 depicts the central region of particles from Figure 2.11 divided into a 6 by 8 matrix of interrogation boxes. Also shown is a sample interrogation box which has been derotated and then "overlapped" with the original image. Finally, Fig 2.12 contains a surface map representation of the cross-correlation output for the sample box. The location of the central peak gives the desired $\Delta x, \Delta z$ to compute the velocity vector. This vector is indicated on the sample box, and aids in the visual identification of 4 to 5 obvious particle pairs. The complete vector field for Figure 2.11 is shown in Figure 2.13. As in the particle tracking method, the peak of the cross-correlation is determined using the intensity centroid technique so the same ±6% uncertainty applies to these vectors.

A source of potentially greater uncertainty arises when the image is tainted with stray background light (noise). When this is the case (especially near the side walls) the image may look like Figure 2.14. Here the cross-correlation map from a sample interrogation box shows several peaks rather than a single central peak. Even with the selected vector shown on the sample box overlay in Figure 2.14, it is more difficult to identify particle pairs here than it was in Figure 2.12. The high density method uses the same techniques to isolate the "real" peak. The range of velocity and angle are limited to within expected bounds (typically $-5<w<+5$m/s and $1<u<7$m/s for a mean $|V|$ flow of 3-4 m/s). Also, vectors in neighboring boxes are consulted to refine the peak selection. And, as a further noise filter, a candidate peak is constrained to have a magnitude >50% of the maximum peak in the correlation to be considered for selection. Unfortunately, in some cases the peak must still be hand picked based on user's judgment or the vector deleted altogether from the image. This of course adds uncertainty to the measured velocity.

To better understand the source of this uncertainty, a simulation was written to generate artificial PIV images with user selected flow conditions. Variables include: particle density, rotational frequency, pulse time separation, mean Δx particle motion, mean Δz particle motion, turbulence level ($\Delta x'/\Delta x_{mean}$ and $\Delta z'/\Delta z_{mean}$), x and z gradients in Δx_{mean} and Δz_{mean} , variable particle intensity, and particle drop-out rate. These last two, particle drop-out and variable intensity, simulate real-life limitations of the PIV technique.

Since the laser sheet is of finite thickness and has a gaussian intensity distribution, any velocity component normal to the sheet (in the y direction) will cause a percentage of particles to drop outside the image before the second laser pulse. At the same time, some new particles will drop into the image that were not present during the first pulse. Finally, even if a particle remains in the sheet for the two laser bursts, its signature may be more or less intense with the second pulse. All of these features create noise (erroneous peaks) in the correlation (e.g. Figure 2.14).

Figure 2.15 shows a fabricated image with 0% drop-out and low random motion (turbulence). Also shown is a sample box overlay and associated correlation map. The vector indicated on the sample box overlay aids identification of numerous particle pairs. Figure 2.16 is another fabricated image with 50% drop out and high turbulence (but the same Δx_{mean} and Δz_{mean} etc...). Here the vector indicated on the sample overlay box does not have numerous associated pairs. Comparing the correlation maps for Figures 2.15 and 2.16, the one with no drop-out and low turbulence has a single dominant peak. The same peak in the case with 50% drop-out and high turbulence is greatly reduced in magnitude. Also, a number of "erroneous" peaks of comparable magnitude obscure the correct peak selection in Figure 2.16.

Returning again to the correlation maps from actual PIV data (Figures 2.12 and 2.14), many similarities with the fabricated data are evident. The high drop-out fabricated data correlation map (Figure 2.16) is similar to the data correlation map near the wall (Figure 2.14). While the 0% drop-out fabricated data correlation map (Figure 2.15) resembles the centerline data correlation map (Figure 2.12). This suggests that the difficulties associated with selecting the correct vector in near wall PIV images stem from particle drop-out and high turbulence. This is not surprising since we expect the flow to be highly three dimensional in this region, with large scale vortical structures and high levels of turbulence. While particle drop-out can be minimized by reducing the pulse separation, there is a trade off with the accuracy of the velocity measurement. As the pixel separation between particles is reduced relative to the uncertainty in the correlation peak determination (constant at <0.5 pixel) the corresponding velocity uncertainty increases.

Steep velocity gradients are another source of uncertainty near the side walls. Adrian [1] recommends $|\Delta u|/u < 0.2$ in an interrogation region to avoid inaccuracies due to low velocity bias and correlations with multiple erroneous peaks. Low velocity bias occurs because particles with a lower velocity have a greater propensity to remain in the sheet for the second pulse. So, any correlation will be biased toward the particles of lowest velocity. In the near wall data presented Figure 2.14 and chapter 4, $|\Delta w|/u$ can reach 0.5 within a y/d distance comparable to the laser sheet thickness (0.4mm). So, the image processing will produce vectors with a wide range of u and w because the flow features are spatially of the same order as the light sheet thickness. Figure 2.17 is the processed vector field from the image in figure 2.14. There is a wide range of u and w since the image was taken at y/d=0.93. Data presented in chapter 4 will show that y/d=0.93 is the Coriolis vortex center for this test condition. For these reasons, the data presented in chapter 4 uses primarily statistical and ensemble averages to describe a flowfield that is really very unsteady and complex.

It is difficult to know how to adequately express the uncertainty associated with these practical limitations. The uncertainty is a function of y/d since the optical noise and flow volatility increase markedly toward the side walls (y/d = 0 & 1). One measure of the uncertainty associated with various images is the percentage of interrogation boxes requiring manual selection of the correlation peak. A second is the range of mean image velocities at a given y/d. Both of these representations are shown in Figure 2.18 for an entire set of data. The variance grows dramatically near the wall. Another representation of the randomness in the data is found by taking the standard deviation of all of the vectors in each image plotted as a function of y/d (Figure 2.19). Again, the range of values grows substantially near the wall. Part of this is due to the fluid unsteadiness and steep velocity gradients, and part is due to the uncertainty of selecting the "correct" peak in a noisy correlation map.

In summary, the error sources are well documented and amount to 6% for each vector. In addition there is an image bias error of 1% and a bias error for entire sets of images of 4% (due to the pixel center of rotation determination). Finally, because the Coriolis vortex is confined to a region near the side wall comparable to the laser sheet thickness, surface reflections, particle dropout, high turbulence, and steep velocity gradients combine to increase the velocity uncertainty here (by 10-15%). Again, this is why the reporting in chapter 4 focuses on mean and ensemble-averaged velocity data rather than isolated vectors.

Wall Temperature Measurements

The wall temperature measurement is made using an Electro-Optical System HgCdTe infrared detector [2]. The detector is housed in a dewar of liquid nitrogen which is mounted on a two-axis traverse system with the focusing optics (Fig 2.1 & 2.20). The magnification of the convex/concave 2 mirror focus system is approximately 1, so the spatial resolution of the device is equal to the size of the HgCdTe sensor, which is 1mm^2.

This is a considerable improvement over traditional temperature measurement using thermocouples, which typically provide only 4-6 measurement locations along each side of a 14d passage [51]. Guidez [16] is the only other researcher to have utilized this high spatial resolution measurement technique. Similar resolution has more recently been attained by using liquid crystals [14]. Two flat mirrors are located at 90° to each other behind the test section and provide optical access to all four sides of the rotating passage (Figure 2.21). The horizontal traverse is used to move the focal point from one side to another and the vertical traverse allows measurement along the axis (x direction) of the passage.

To acquire a signal, the sensor and optics are moved to a desired location and remain fixed while the heated test section is rotated in front of them. When the test section passes into the focal point of the optics, the radiation emitted from the heated surface is focused onto the sensor. The sensor produces a voltage signal proportional to the level of incident radiation. Then using the Stefan-Boltzmann Law, $q_{rad} \propto T^4$, the temperature of this spot on the surface can be deduced. The AC coupled sensor has a response time on the order of microseconds which means that an entire row of data can be collected during a single pass of the test section. This data then represents the temperatures in a 1mm wide strip on one side of the test section, at a given x location. Obviously, the passage must be rotating to obtain a signal (since the IR sensor is stationary), so a Rot=0 reference data set can only be approached. Previous researchers who used the same facility have described in detail the process of focusing, calibrating, and testing the IR system [2,29,15], so only a brief outline is included here.

Using optical development software the focal spot size ($1mm^2$) of the double mirror optical system was found to increase by a factor of >4 for a focal shift of 5mm (Figure 2.22). To insure a spatial resolution within +20% of the nominal value, the best focus position must be maintained to within ±1mm. This is done for each side by observing the sharpness of the ITO/quartz and quartz/air interfaces during a "focusing" run before testing. Since quartz has a factor of 3 greater emissivity the signal here is much stronger for the same temperature. By translating the optical system horizontally while monitoring the IR signal each time the heated passage rotates in front of it, the point of best focus (sharpest interface resolution) is determined for each side.

Once the focal positions are determined, a calibration procedure is employed as follows. A copper bar with square cross-section just smaller than the 10mm x 10mm ID of the quartz passage is inserted into the test section extending from inlet to exit. A resistive heating element and three Type K thermocouples (spread evenly along its length) are embedded in the copper bar. Once heated, the copper creates a uniform temperature distribution with temperature gradients primarily in the axial direction due to conduction out the ends of the tube. With time, these gradients become small and the entire quartz passage assumes the temperature of the calibration bar. The thermocouples provide 3 reference temperatures which are fit with a parabola to give T(x) along the copper bar (and by assumption along the quartz exterior wall also). Infrared maps for all four sides are then taken at 3-5 different temperature (heat flux) settings. The sensor output voltage data (V_{dc}) from each spatial location is then collected and correlated with a linear fit.

$$T^4 = C0 + C1 \cdot V_{dc}$$

The result is a C0 and C1 coefficient for each spatial location. These are then used to convert subsequent voltage measurements from the IR sensor to temperatures:

$$T = \left(C0 + C1 \cdot V_{dc}\right)^{1/4}$$

Performing a calibration in this manner eliminates the temperature measurement sensitivity to local variations in emissivity and view factor since each spot on the surface is individually calibrated (no emissivity or view factor is assumed). The mean error in the above linear fit is 2°C for the temperature measurement. Figure 2.23 shows a sample temperature map from an actual test run.

Nusselt Calculation

To compute the Nusselt number on the inside of the test section both the local temperature and heat flux must be determined. The Nu is calculated by,

$$Nu = \frac{q/(T_w - T_{film})}{k_{film}/d}$$

where $T_{film} = (T_w + T_{fluid})/2$ and the thermal conductivity, k, is evaluated at the film temperature. The available data, though, is not on the interior walls, rather it is on the exterior walls of the passage. So, several steps are required to move from the available data to the desired Nu. This section reviews these necessary steps.

First, to obtain the distribution of heat flux, q, generated in the ITO film, the current in each of the four sides is deduced from the parallel current law. The film resistances measured at room temperature were: front side wall = 50.5Ω, leading wall = 60.5 Ω, back side wall = 69 Ω, and trailing wall = 61.5 Ω. The net resistance is 14.9 Ω. To determine the resistance at the operating conditions, these side resistances are adjusted to reflect an average operating temperature higher than room temperature. This adjustment is computed using the temperature coefficient of resistivity α = 310e-6/°C described earlier in this chapter. From this the current and potential drop on each side is calculated. The potential drop and surface temperature map are used to operate the variable resistivity code mentioned earlier (Appendix B). Again, the code adjusts the local heat flux for variations in the current density due to the dependence of local resistivity on temperature. The result is nearly constant heat flux for each side (α is low enough for ITO that it has <1% effect on q for these test conditions).

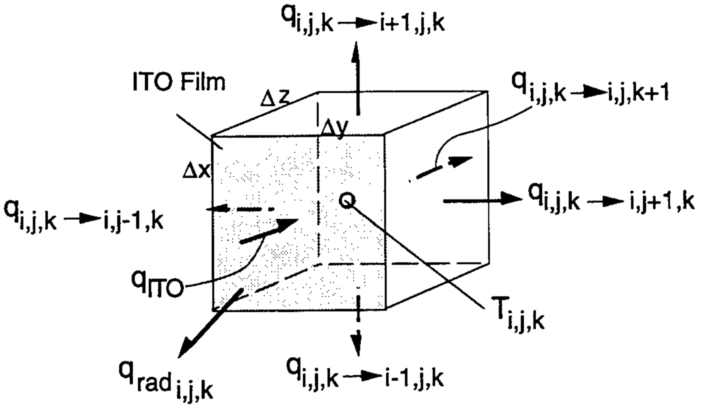
The next step is to assess the heat loss due to radiation from the passage exterior walls to the surrounding vacuum chamber walls. This is estimated using,

$$q_{\text{rad}} = \sigma\varepsilon_{\text{ITO}}\left(T_{\text{ITO}}^4 - T_{\text{enclosure}}^4\right)$$

where $\varepsilon_{\text{ITO}} = 0.27$. This component is approximately 4.5% of the total heat flux generated in the film. Because it is a function of temperature, hot spots have a larger radiation than cold spots. This introduces an additional spatial non-uniformity of ±1.5%. With this q_{rad} deducted from the q generated in the ITO, the remainder is conducted into the passage and ultimately into the coolant flow. The last step is then the conduction through the quartz. With a wall thickness/passage width ratio of 1.5mm/10mm = 15% the passage is not thin enough to ignore lateral conduction and simply assume all conduction is 1-D. So, a finite element analysis of the heat flow was undertaken to convert T and q on the exterior surface to the desired T and q on the interior surface. Applying conservation of energy to an elemental volume at the test section exterior surface, and assuming steady state,

$$q_{\text{ITO}}\Delta x\Delta y = q_{\text{rad}_{i,j,k}}\Delta x\Delta y + q_{i,j,k\to i,j+1,k}\Delta x\Delta z + q_{i,j,k\to i,j-1,k}\Delta x\Delta z$$

$$+q_{i,j,k\to i+1,j,k}\Delta y\Delta z + q_{i,j,k\to i-1,j,k}\Delta y\Delta z + q_{i,j,k\to i,j,k+1}\Delta x\Delta y$$


$q_{i,j,k \rightarrow i+1,j,k}$
ITO Film
Δz
Δy
$q_{i,j,k \rightarrow i,j,k+1}$
Δx
$q_{i,j,k \rightarrow i,j+1,k}$
$q_{i,j,k \rightarrow i,j-1,k}$
q_{ITO}
$T_{i,j,k}$
$q_{rad\, i,j,k}$
$q_{i,j,k \rightarrow i-1,j,k}$

Since the ITO film thickness (4500 Angstroms) is much smaller than the quartz passage wall thickness (1.5mm), the heat generation in the film is essentially a uniform surface heat flux. Then, by applying Fourrier's law of steady heat conduction,

$$q = -k\frac{dT}{dx}$$

the heat flows (q) between adjacent elements of the passage wall can be expressed in terms of temperature differences. Discretizing Fourrier's law yields,

$$q_{i,j,k \rightarrow i+1,j,k} = -k\frac{T_{i+1,j,k} - T_{i,j,k}}{\Delta x}$$

Then substituting this into the conservation of energy equation, we have

$$q_{ITO}\Delta x\Delta y = \sigma\varepsilon(T_{i,j,k}^4 - T_{enclosure}^4)\Delta x\Delta y + k\frac{T_{i,j,k} - T_{i,j+1,k}}{\Delta y}\Delta x\Delta z + k\frac{T_{i,j,k} - T_{i,j-1,k}}{\Delta y}\Delta x\Delta z$$

$$+k\frac{T_{i,j,k} - T_{i+1,j,k}}{\Delta x}\Delta y\Delta z + k\frac{T_{i,j,k} - T_{i-1,j,k}}{\Delta x}\Delta y\Delta z + k\frac{T_{i,j,k} - T_{i,j,k+1}}{\Delta z}\Delta x\Delta y$$

Since quartz is opaque at the IR wavelength, the radiometer measures the temperature on the exterior surface. So, all of the temperatures at "k" are known. Solving for the only unknown temperature, $T_{i,j,k+1}$:

$$T_{i,j,k+1} = T_{i,j,k} - \frac{q_{ITO}}{k}\Delta z + \frac{\sigma\varepsilon}{k}\left(T_{i,j,k}^4 - T_{enclosure}^4\right)\Delta z + \left(T_{i,j,k} - T_{i,j-1,k}\right)\left[\frac{\Delta z}{\Delta y}\right]^2$$
$$+\left(T_{i,j,k} - T_{i,j-1,k}\right)\left[\frac{\Delta z}{\Delta y}\right]^2 + \left(T_{i,j,k} - T_{i+1,j,k}\right)\left[\frac{\Delta z}{\Delta x}\right]^2 + \left(T_{i,j,k} - T_{i-1,j,k}\right)\left[\frac{\Delta z}{\Delta x}\right]^2$$

Once all the element temperatures at "k+1" are determined, a similar expression for the temperature at "k+2" in terms of "k+1" values is solved. When the interior surface element is reached, its temperature and exiting heat flux allow the calculation of the convective heat transfer coefficient, h, and Nu=hd/k.

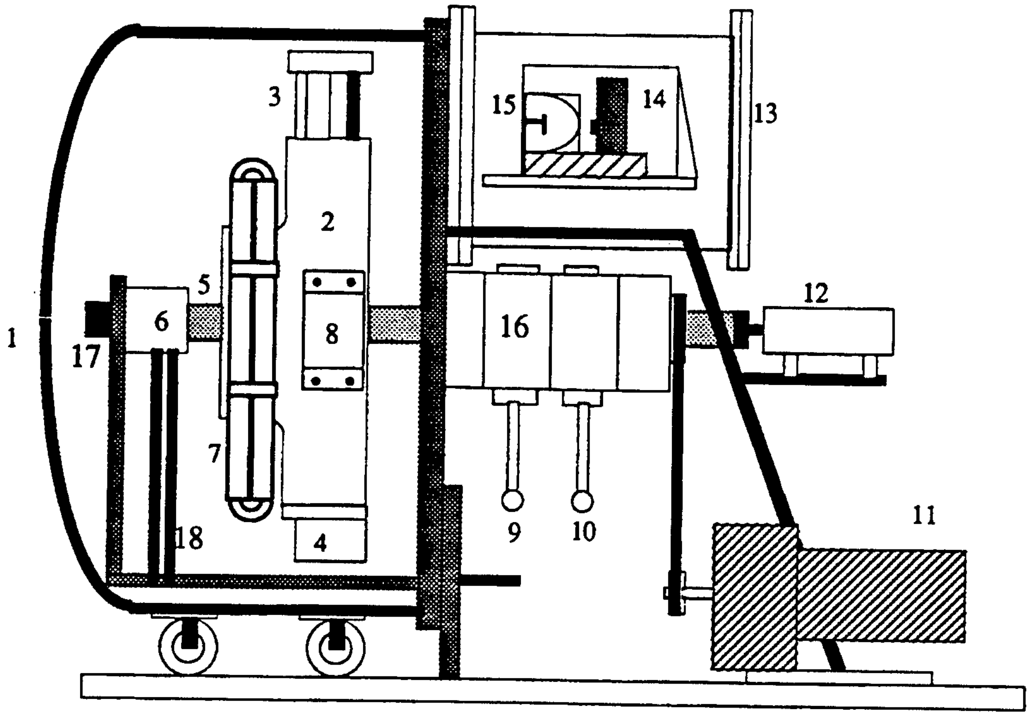
$$h = \frac{q_{i,j,k+n \to i,j,k+n+1}}{T_{i,j,k+n} - T_{film}}$$

This straightforward implementation produces the result that while all the temperatures drop through the thickness, hot spots experience less of a drop than cold spots. This is because a hot spot dumps a higher percentage of its available heat flux laterally and has less heat available to conduct through the thickness. What results is a non-negligible readjustment of q away from hot spots in the passage wall. So, when compared to a simple 1-D conduction analysis, the Nu calculated using a 3D conduction algorithm is lower at the hot spots and higher at the cold spots. The 3D algorithm also incorporates the variation in quartz' thermal conductivity, k, with temperature which results in an additional ±2.5% variation in q (better conduction at hot spots). Figure 2.24 compares internal surface temperature maps obtained using both 1D and 3D conduction. The difference is barely perceptible here. The bulk of the effect is seen in the q distribution, which is shown in Fig 2.25 for both 1D and 3D. Here the average q for each side is plotted as a function of x. The difference between the two methods of conduction analysis is larger than 10% at some points. Medwell et al [36] also found that accounting for wall conduction was essential to correctly predicting Nu in their computational study of a circular cross-section passage with constant heat flux on the exterior wall. Figure 2.25 also portrays the disparity in heat flux generated from the four ITO films. As mentioned earlier, the bulk resistances are not equal (owing to film manufacturing variations larger than expected), and since they are connected in parallel, the generated heat flux also varies. The effect on the data is reviewed in chapter 4.

One final note regards the sensitivity of the finite element algorithm to discontinuous temperature or heat flux distributions on the exterior surface. Because such small temperature differences drive a large heat flux (ΔT=1°C generates 600 W/m^2 of conducted heat flux, or 6% of the total amount generated in the ITO), an input temperature data map like the raw data in Figure 2.23 requires a dense spatial grid to resolve the conduction without "blowing up". To avoid the computational time required to do this, the data was fit with empirical relations to create smooth, continuous distributions of T and q while preserving the general trends in the data. Hence the even contours in Figure 2.24 (which is an empirical fit) vs. Figure 2.23 (which is raw data). The fit involves a 6th order polynomial fit to the perimeter averaged T(x) data and a single sine wave with variable amplitude and phase to fit the circumferential variations at each x. In addition, the data at the corners of the tube (non-coated) were disregarded, so only the center 10mm of each wall section were used to create a 40mm perimeter in the 3D conduction model. Figure 2.26 compares the raw temperature map to the fitted profile. Even with a smooth temperature profile, the discontinuity in q (from side to side) required a post-smoothing of q and T along a 1mm wide region near each edge of the four ITO-coated strips. The final Nu maps and mean Nu plots are presented in chapter 4.

The net uncertainty in Nu comes from 3 sources: q, T_w , and T_{fluid} . T_{fluid} was determined to within ±1°C using energy conservation along the passage and contributes an uncertainty of 0.7% to Nu. The external T_w measurement has an uncertainty quoted earlier of ±2°C. This contributes ±1.5% uncertainty to Nu. An additional ±1.9% uncertainty in Nu is associated with the empirical temperature fitting (within ±2.5°C) and 3D conduction algorithm used to obtain the internal T_w . Finally, the heat flux measurement has the largest contribution since the current uncertainty of ±2.5% is squared. This results in a q (and Nu) uncertainty of 5%. Using Kline and McClintock's [33] uncertainty estimation methodology, the net absolute uncertainty in Nu is ±9.1%.

For the discussion which follows in chapter 4, it is important to distinguish between relative uncertainty and absolute uncertainty in the Nu calculation. Since many of the conclusions are based on comparisons between data sets, they are subject only to the relative uncertainty. When comparing Nu data from the same side of the test section, the long timescale of the ±2.5% drift in the current supply is not applicable. The only uncertainties that apply are those from the T_w measurement (±1.5%) and 3D conduction (±1.9%). This gives a net relative uncertainty of ±3.4%. When comparing Nu from various sides of the passage, the current drift does apply, and brings the net relative uncertainty in Nu to ±8.4%. Finally, when comparing data from different test runs, the full Nu uncertainty of ±9.1% is applicable. This is evident in the comparison of two sets of Nu data at comparable operating conditions shown in Figure 2.27. The trends in Nu will be fully discussed in chapter 4.



1. Vacuum Chamber / Protective Casing
2. Rotating Arm
3. Test Section and Mirrors
4. Balance Weight
5. Shaft
6. Power Slip Ring
7. Heat Exchangers
8. Onboard Instrumentation Box
9. Inlet flow
10. Outlet flow
11. Motor
12. Instrumentation Slip Ring
13. Housing
14. IR detector
15. Imaging system
16. Seals
17. Optical Encoder
18. Power wires

Figure 2.1: Schematic of Internal Cooling Test Facility

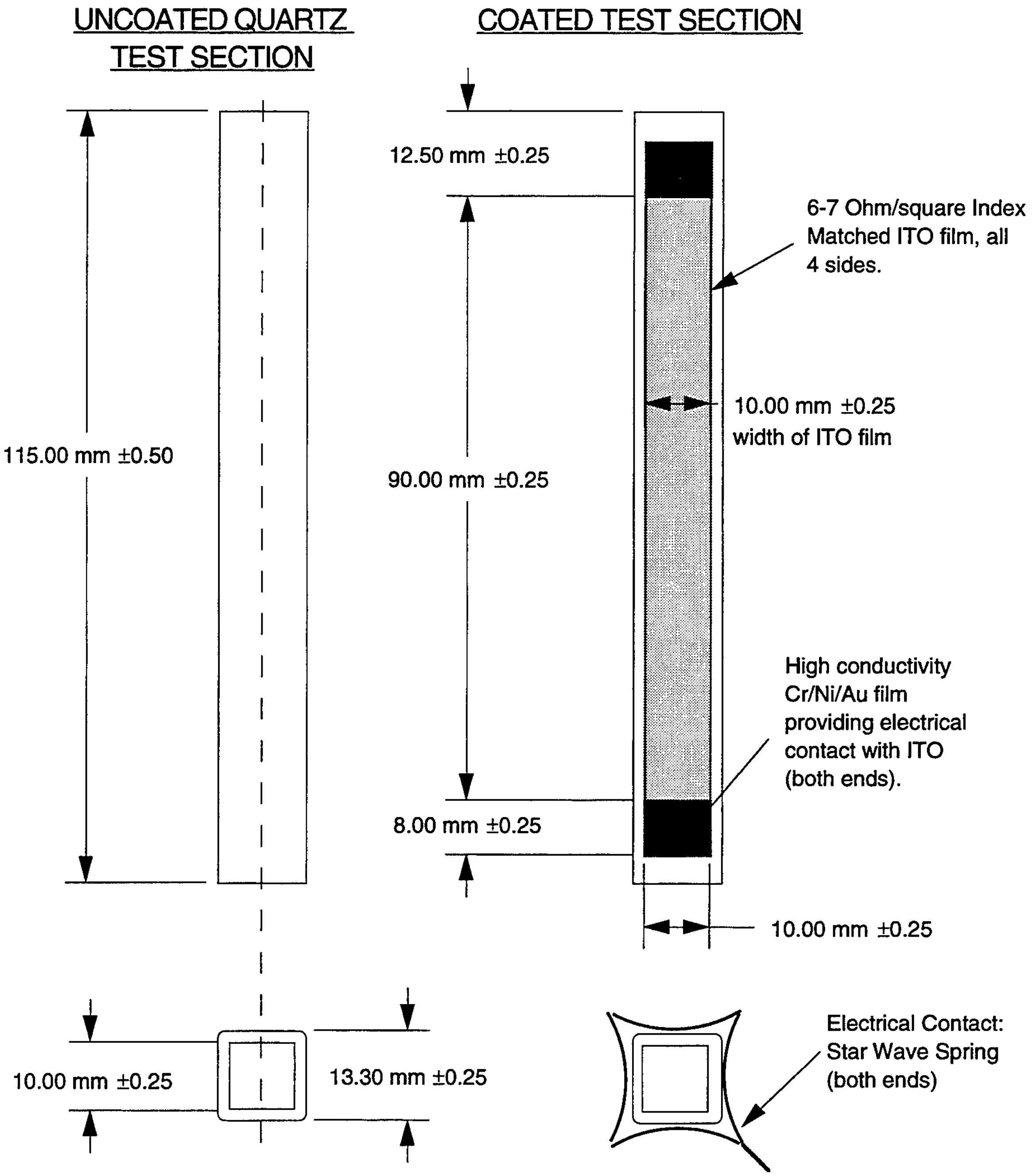


Figure 2.2: Schematic of Quartz Test Section with Indium Tin Oxide (ITO) Coating and Electrical Contact Spring Indicated.

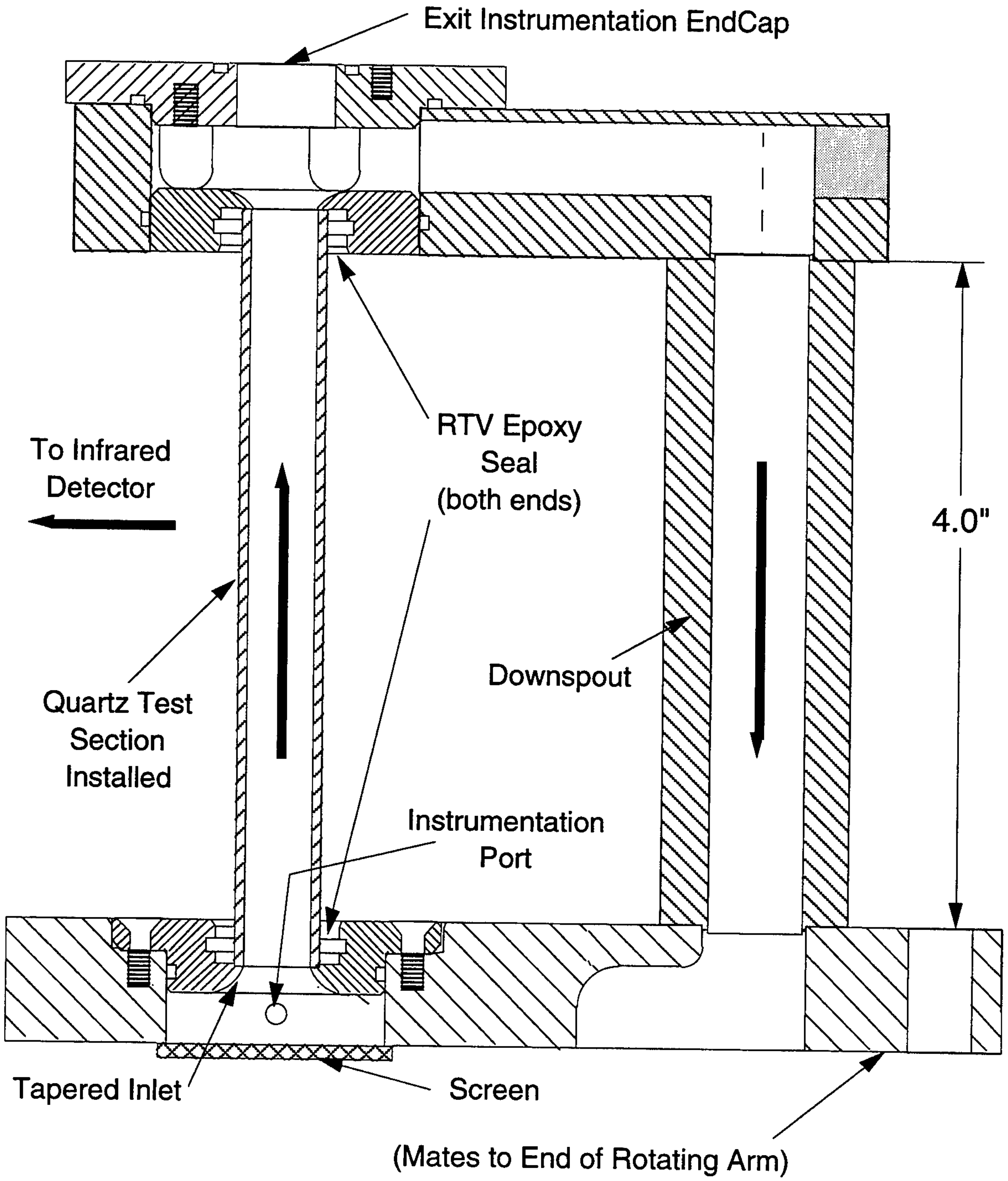

Exit Instrumentation EndCap
To Infrared Detector
RTV Epoxy Seal (both ends)
4.0"
Quartz Test Section Installed
Downspout
Instrumentation Port
Tapered Inlet
Screen
(Mates to End of Rotating Arm)

Figure 2.3: Schematic of Quartz Test Section mounted in Housing on Rotating Arm.

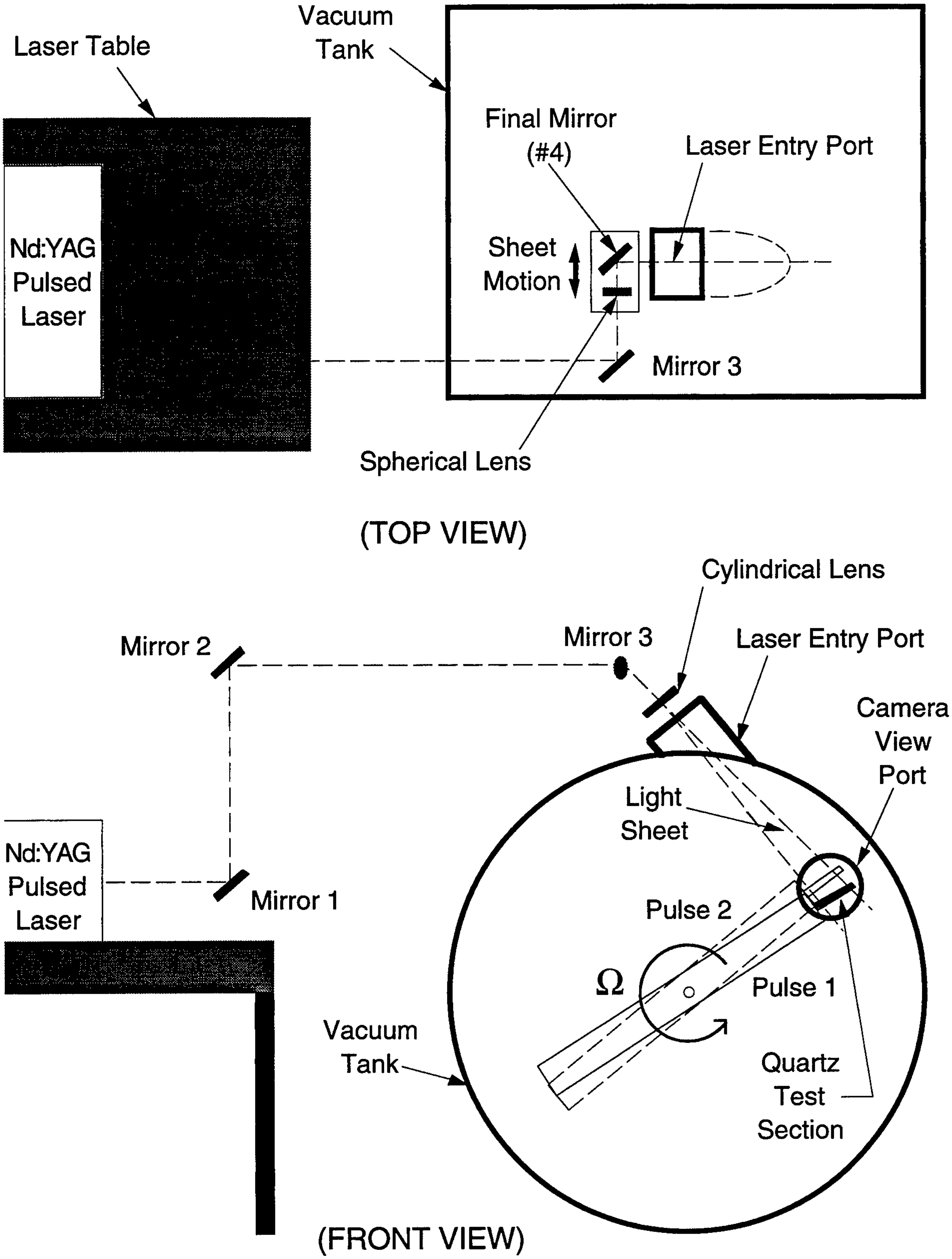


Figure 2.4: Schematic of Laser and Optical Layout.

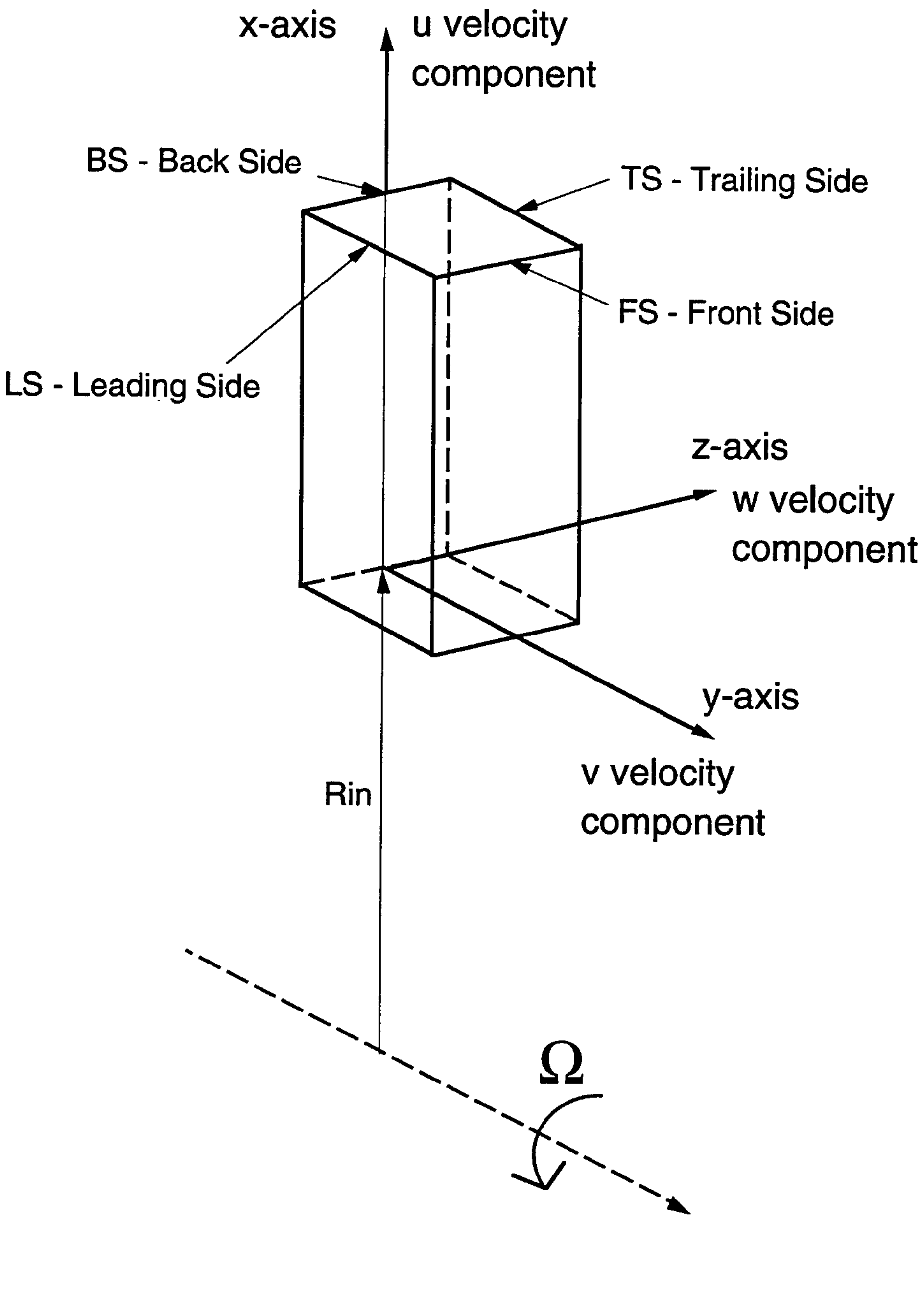


Figure 2.5: Schematic of Test Section showing Axis and Velocity Components.

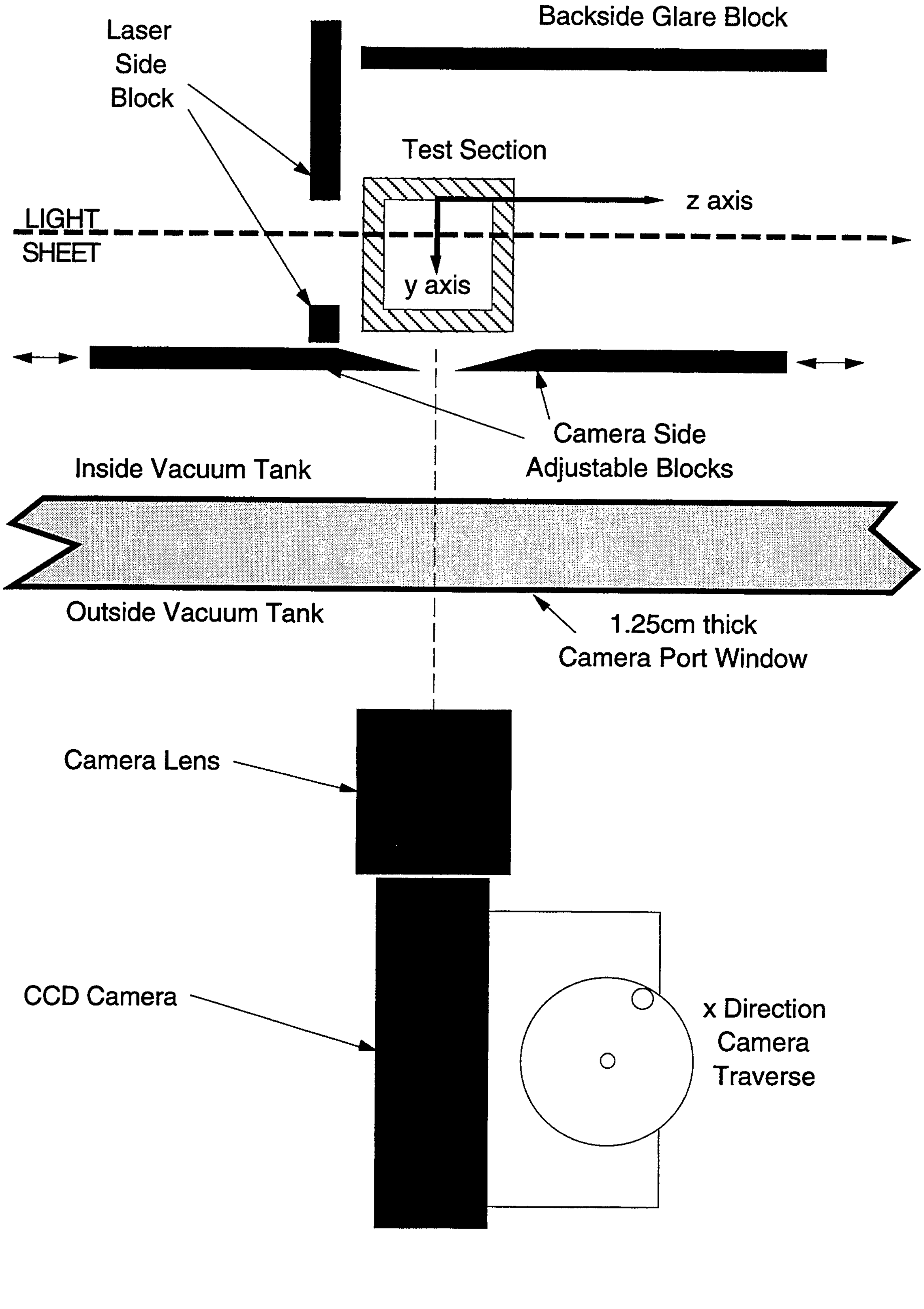


Figure 2.6: Schematic of Camera and Test Section Imaging Layout.

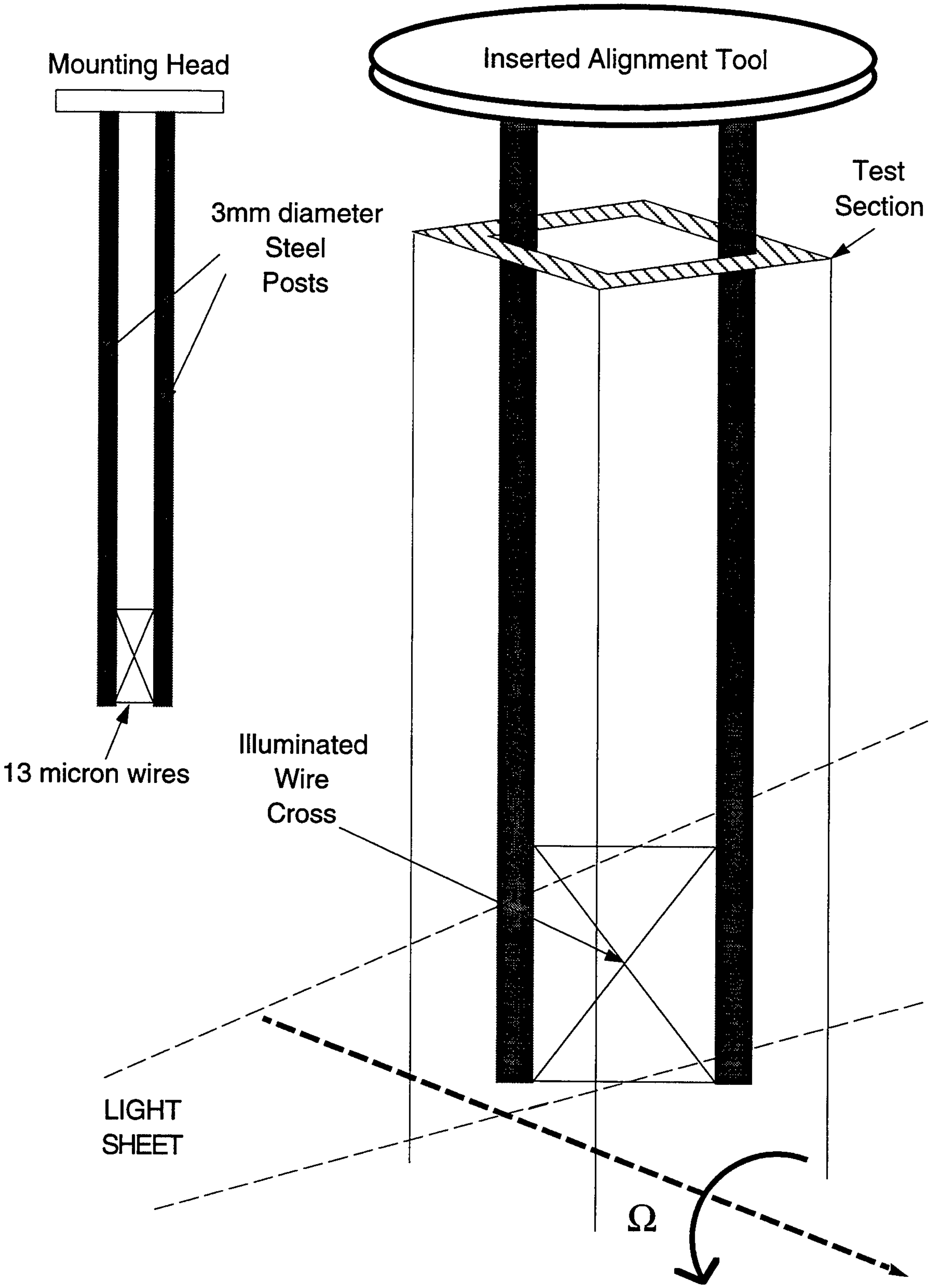
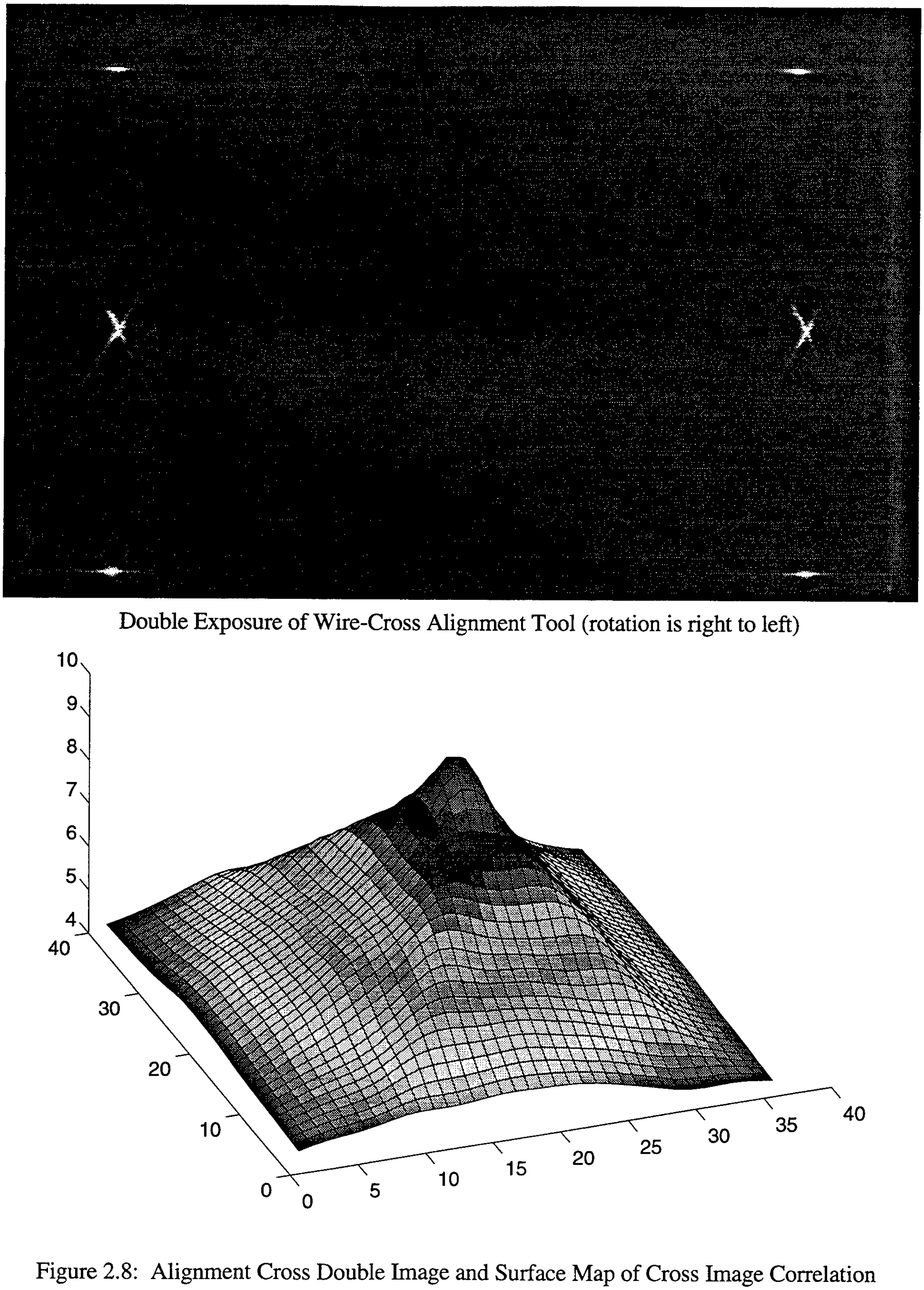


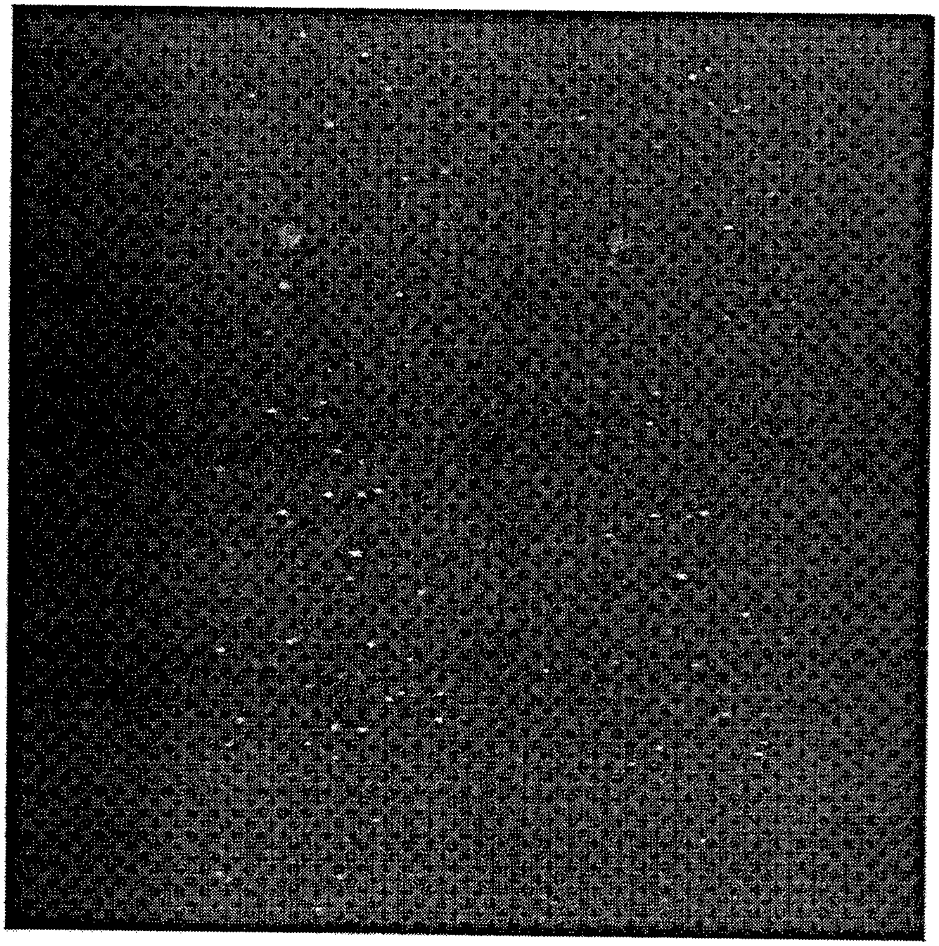
Figure 2.7: Schematic of Laser Alignment Tool Inserted into Rotating Test Section.



Double Exposure of Wire-Cross Alignment Tool (rotation is right to left)

Figure 2.8: Alignment Cross Double Image and Surface Map of Cross Image Correlation

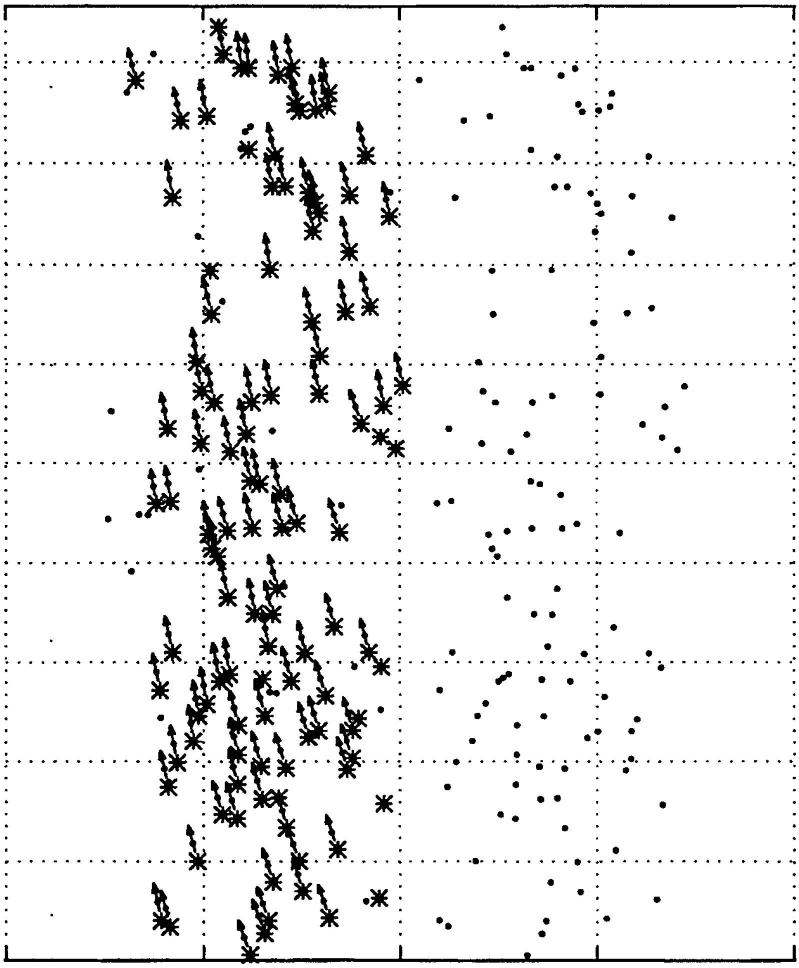
|←——Second Pulse→|←First Pulse——→|



Flow Direction is Bottom to Top. Rotation Direction is Right to Left

Figure 2.9: Sample PIV Image for Low Density Method
Data for Re=2500, Rot=0.45, d.r.=0.0, at x/d=3.8 and y/d=0.09

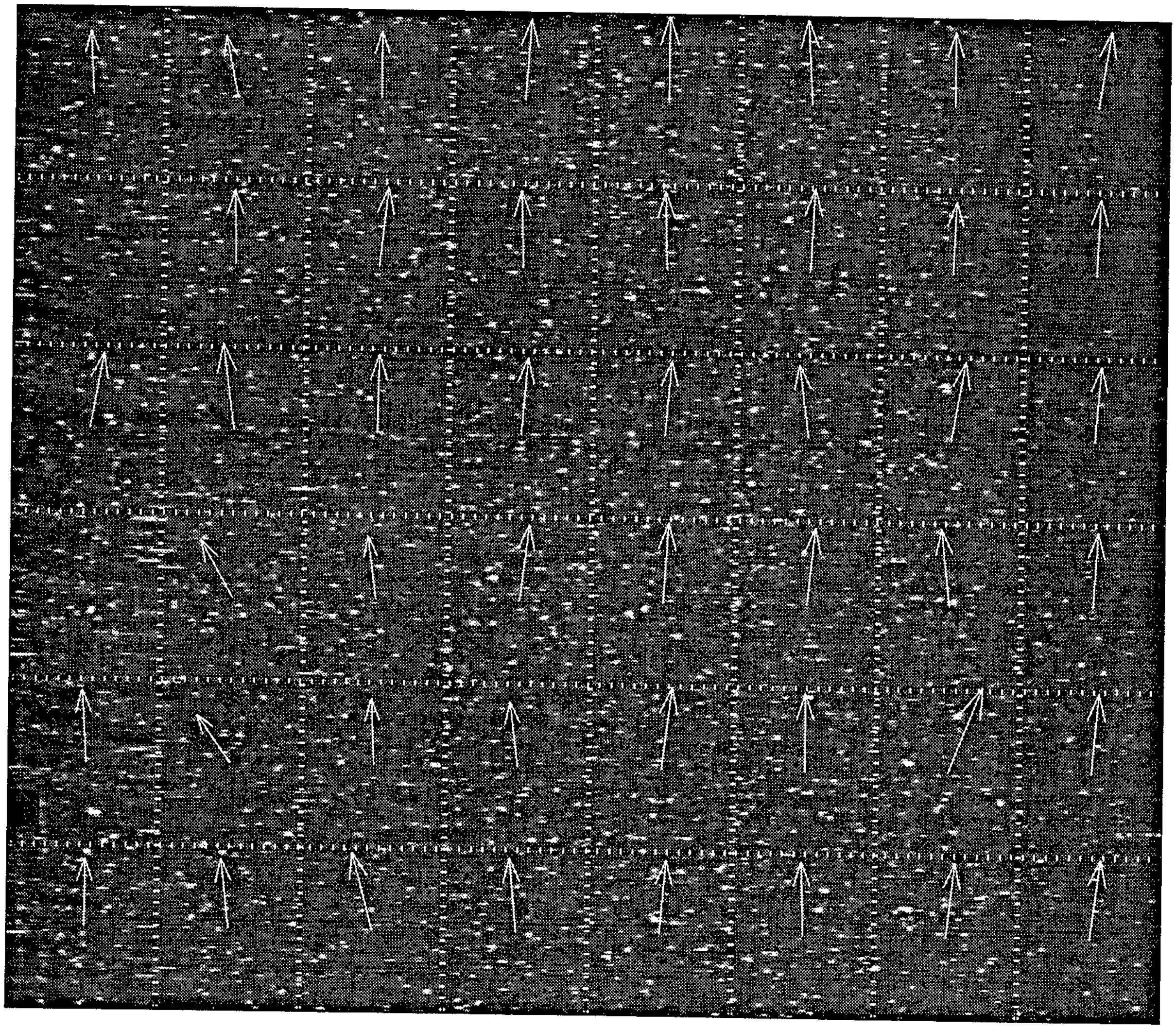
|←—Second Pulse→|←First Pulse—→|



Flow Direction is Bottom to Top. Rotation Direction is Right to Left

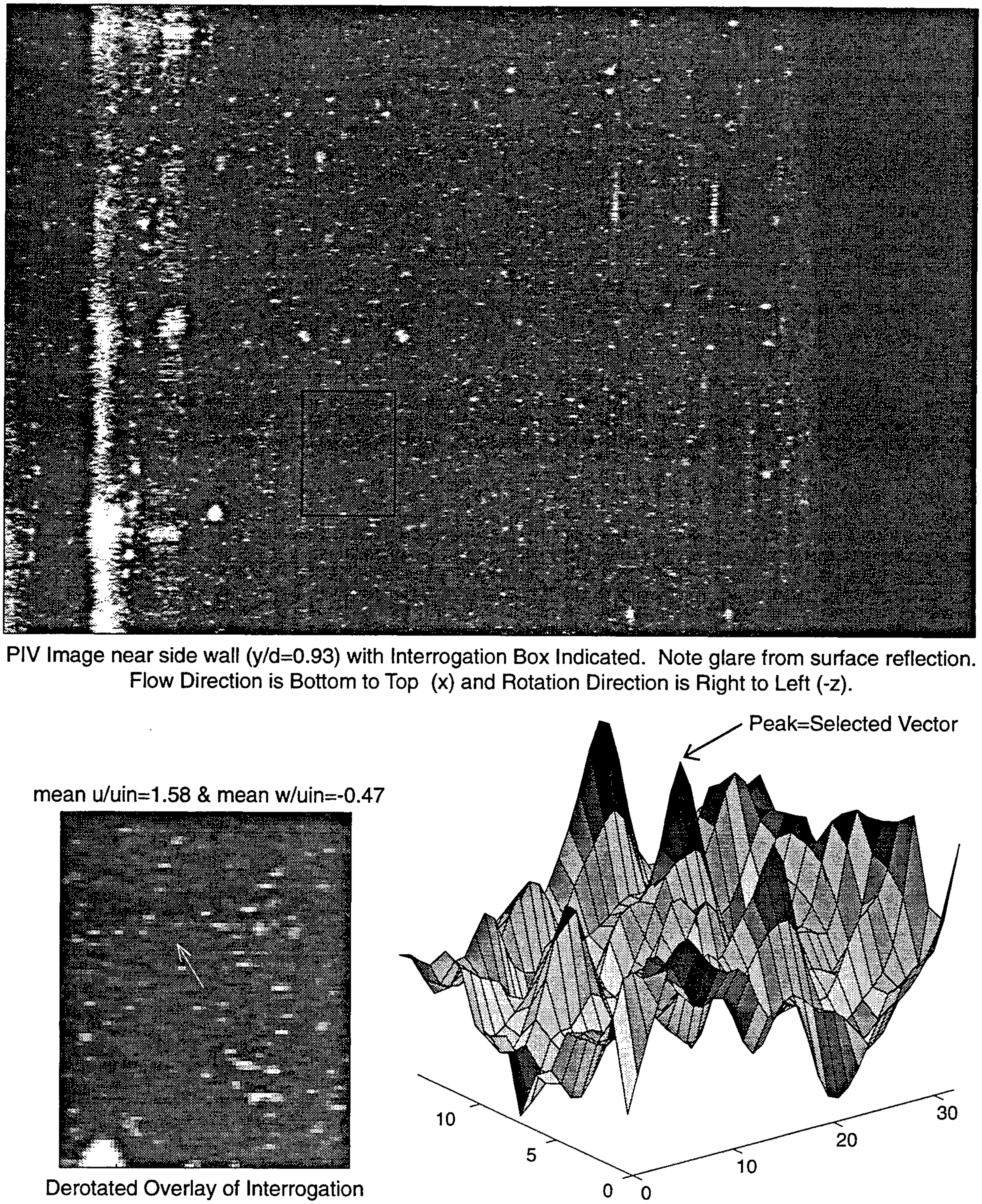
(* indicates "derotated" position of particles from 1st image into 2nd image)

Figure 2.10: Result of Image Processing for Figure 2.9 using Low Density Method
Data for Re=2500, Rot=0.45, d.r.=0.0, at x/d=3.8 and y/d=0.09
Mean Image Vector: u = 1.76 m/s and w=-0.41m/s



Processed PIV Image from Fig 2.11: mean u/uin=1.064 and mean w/uin=0.078 (Vectors Enlarged)
Flow Direction is Bottom to Top (x). Rotation Direction is Right to Left (-z).

Figure 2.13: Processed PIV Image (Figure 2.11) using High Density Method.
Data for Re=8100, Rot=0.2, & d.r.=0.0, at x/d=7.9, y/d=0.44, & -0.3 < z/d < 0.38

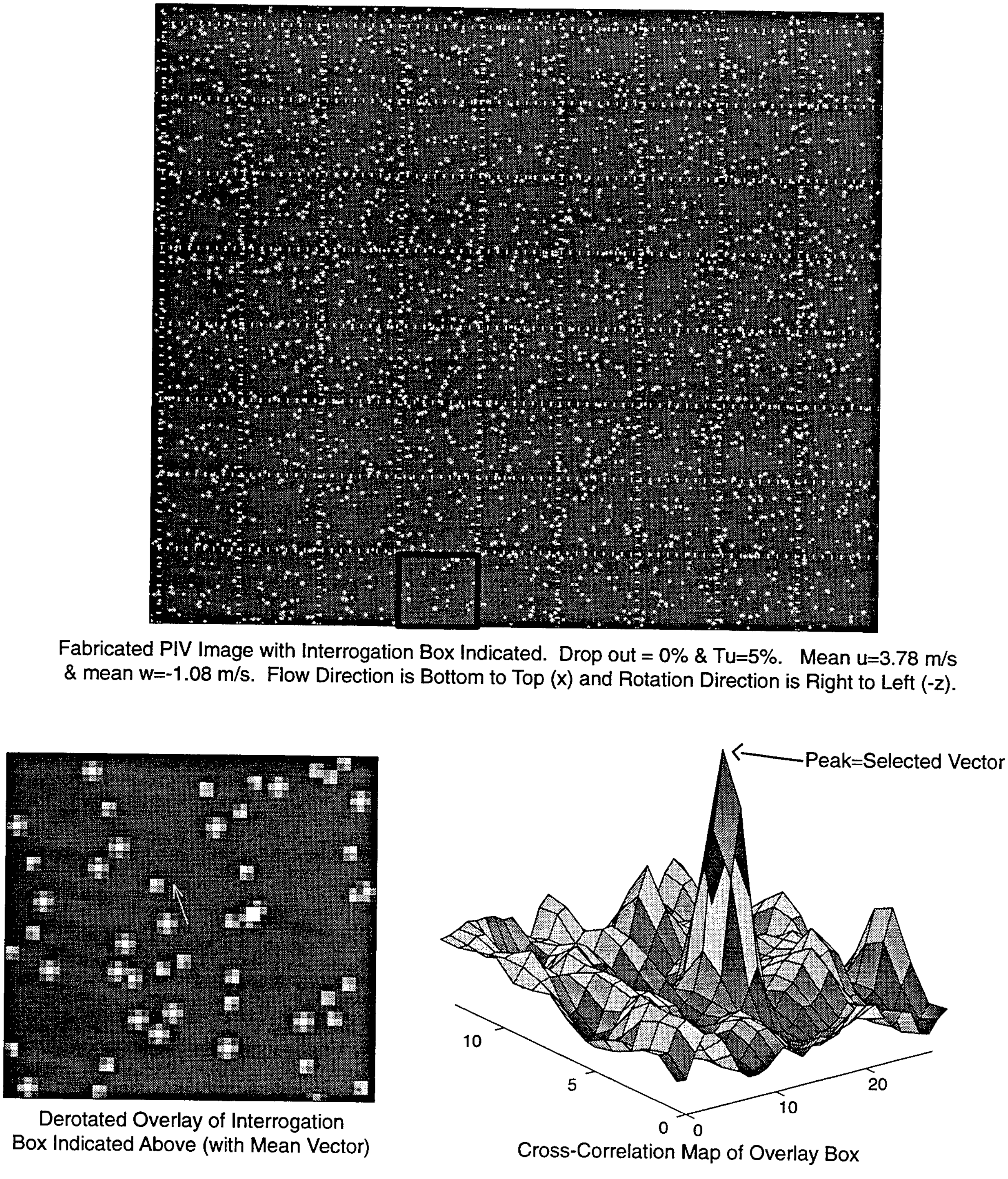


PIV Image near side wall (y/d=0.93) with Interrogation Box Indicated. Note glare from surface reflection. Flow Direction is Bottom to Top (x) and Rotation Direction is Right to Left (-z).

Derotated Overlay of Interrogation Box Indicated Above (with Mean Vector)

Cross-Correlation Map of Overlay Box

Figure 2.14: (a) Sample PIV Image for Re=8100, Rot=0.2, d.r.=0.0, at x/d=7.9 and y/d=0.93 (b) Derotated Sample Box with Overlay and Vector (c) Cross-Correlation Map of Sample Box

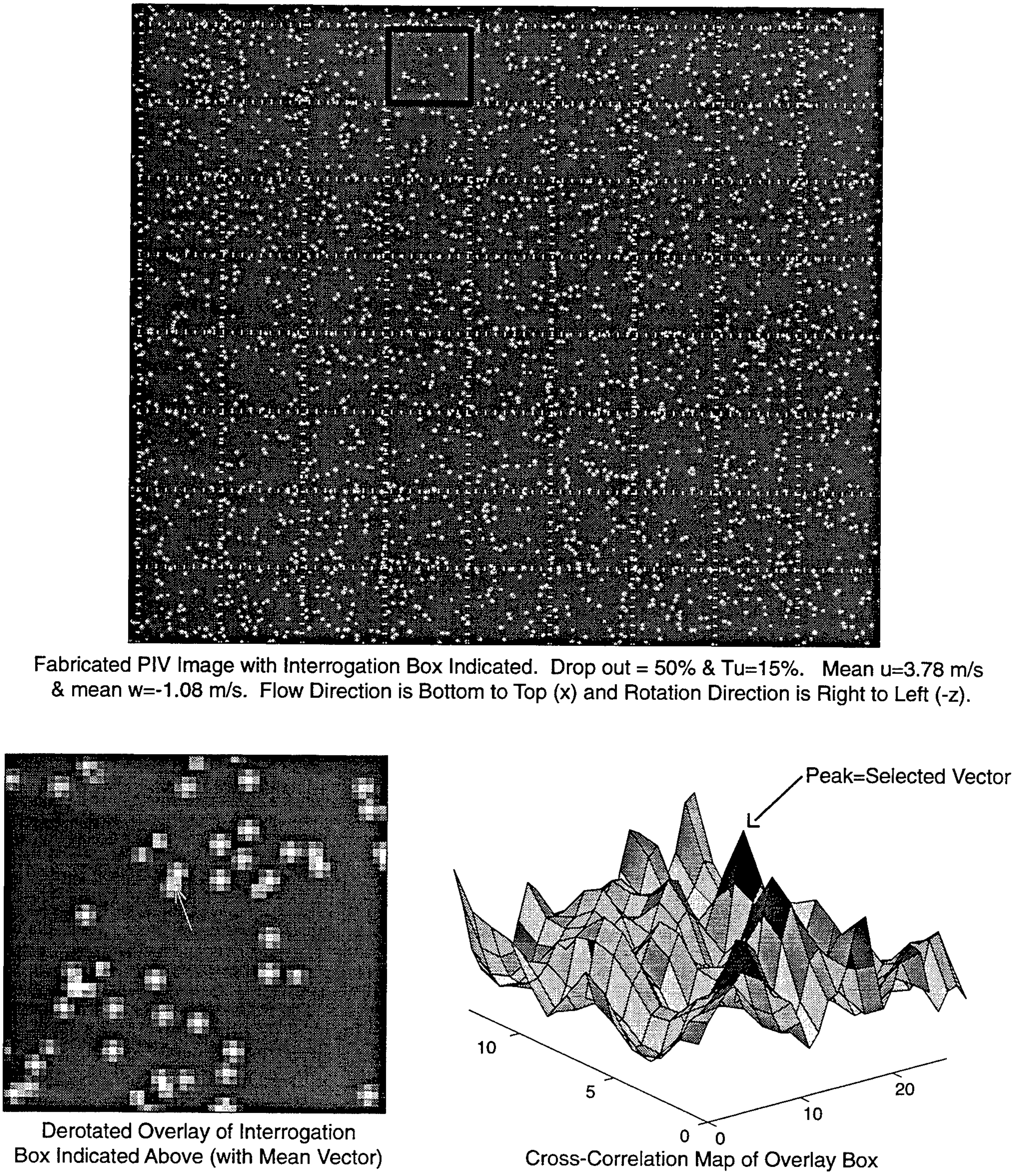


Fabricated PIV Image with Interrogation Box Indicated. Drop out = 0% & Tu=5%. Mean u=3.78 m/s & mean w=-1.08 m/s. Flow Direction is Bottom to Top (x) and Rotation Direction is Right to Left (-z).

Derotated Overlay of Interrogation Box Indicated Above (with Mean Vector)

Cross-Correlation Map of Overlay Box

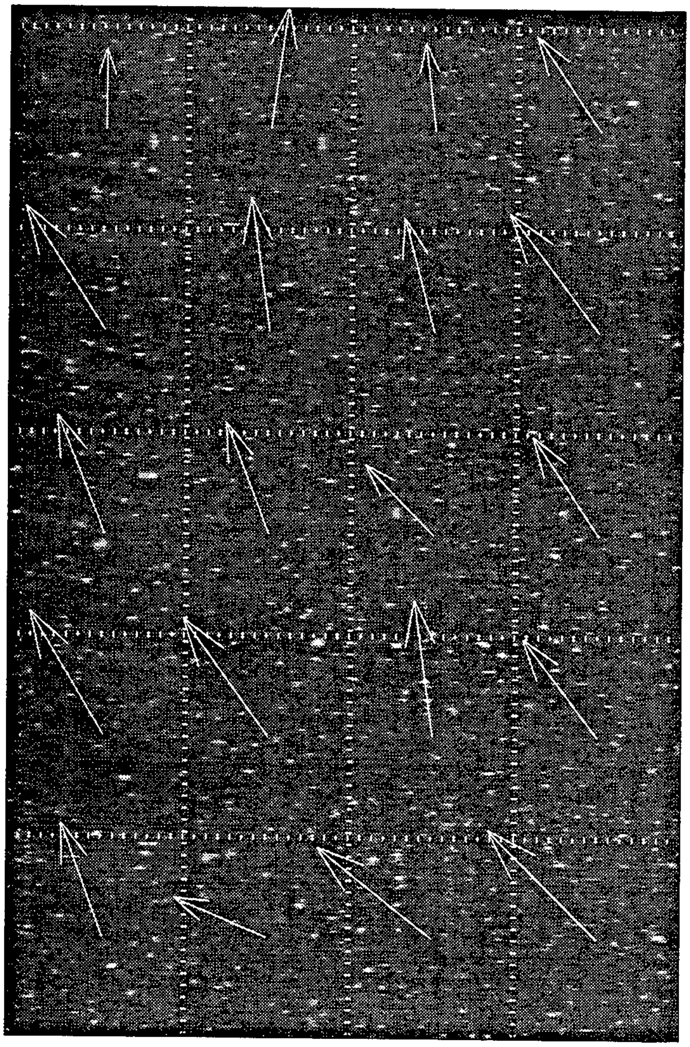
Figure 2.15: (a) Fabricated PIV Image with 0% Particle Drop-Out and 5% Turbulence. (b) Derotated Sample Box with Overlay and Vector (c) Cross-Correlation Map of Sample Box


Fabricated PIV Image with Interrogation Box Indicated. Drop out = 50% & Tu=15%. Mean u=3.78 m/s & mean w=-1.08 m/s. Flow Direction is Bottom to Top (x) and Rotation Direction is Right to Left (-z).

Derotated Overlay of Interrogation Box Indicated Above (with Mean Vector)

Cross-Correlation Map of Overlay Box

Figure 2.16: (a) Fabricated PIV Image with 50% Particle Drop-Out and 15% Turbulence. (b) Derotated Sample Box with Overlay and Vector (c) Cross-Correlation Map of Sample Box



Processed PIV Image from Fig 2.14: mean u/uin=1.46
and mean w/uin=-0.656 (Vectors Enlarged)
Flow Direction is Bottom to Top. Rotation Direction is Right to Left.

Figure 2.17: Processed PIV Image (Figure 2.14) using High Density Method.
Data for Re=8100, Rot=0.2, & d.r.=0.0, at x/d=7.9, y/d=0.93, & -0.15 < z/d < 0.2

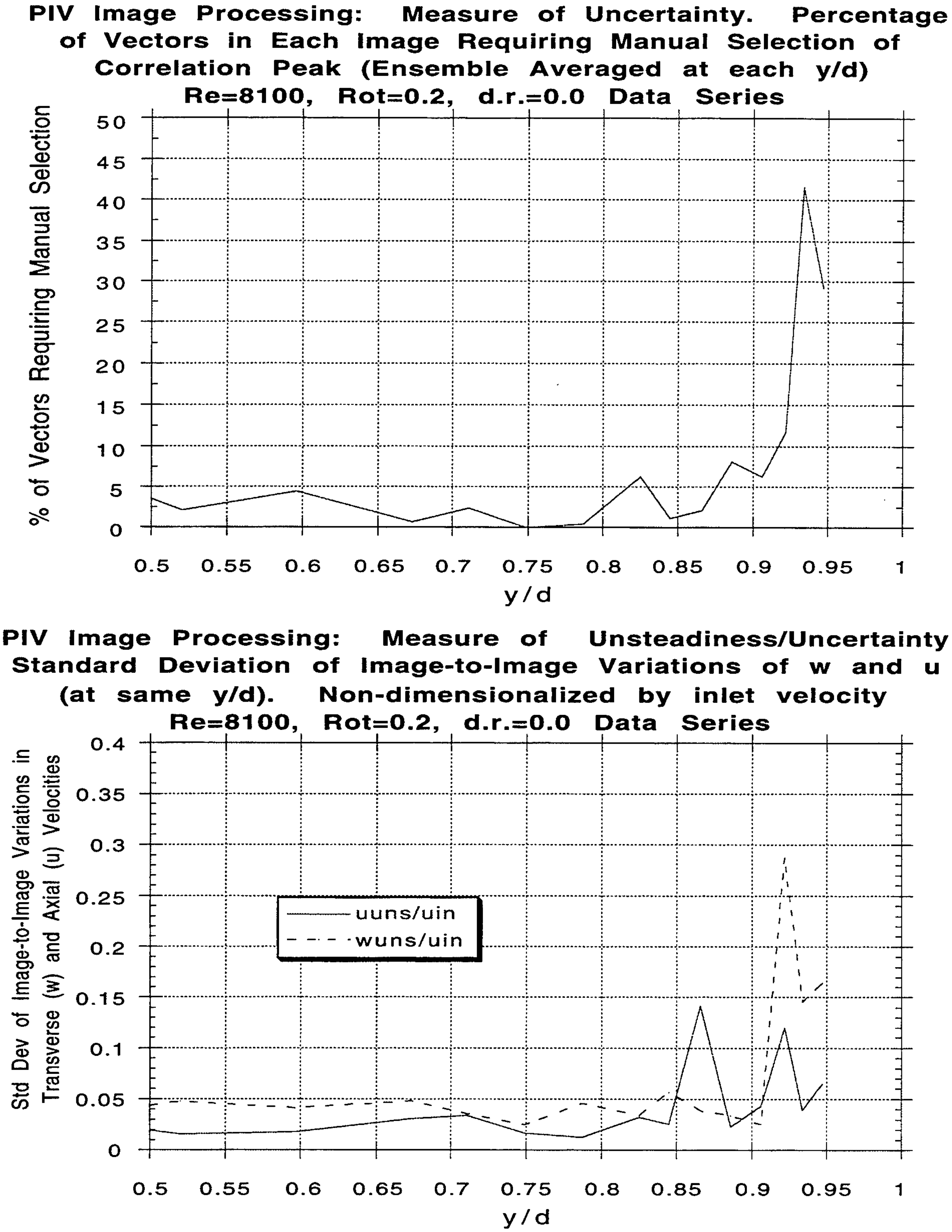


Figure 2.18: Measures of Uncertainty/Unsteadiness in PIV Images. (a) % Vectors Requiring Manual Selection (b) Standard Deviation of Image-to-Image Variations at constant y/d

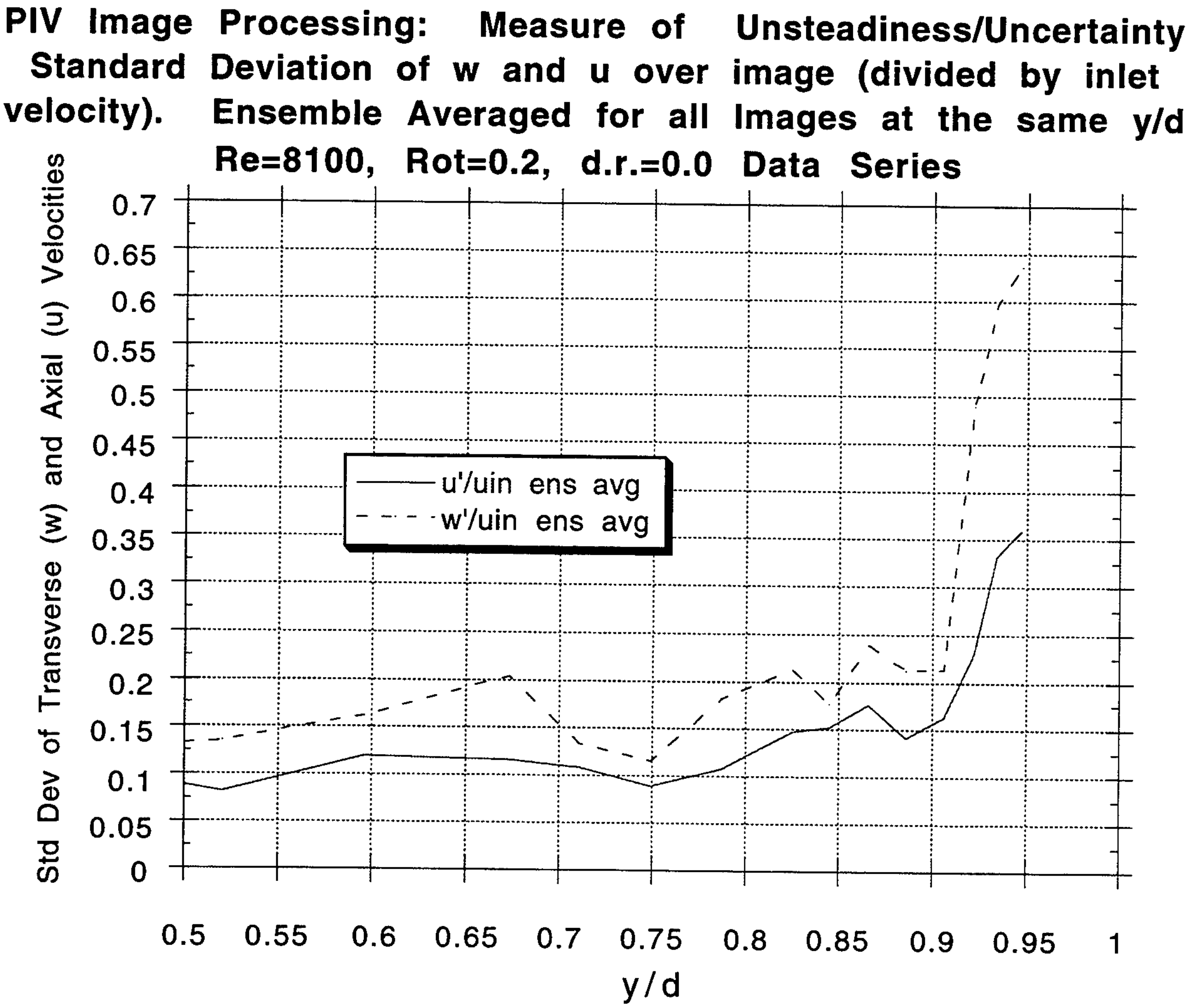


Figure 2.19: Measure of Uncertainty/Unsteadiness in PIV Images.
Ensemble Averaged Standard Deviation of w and u over image at constant y/d.

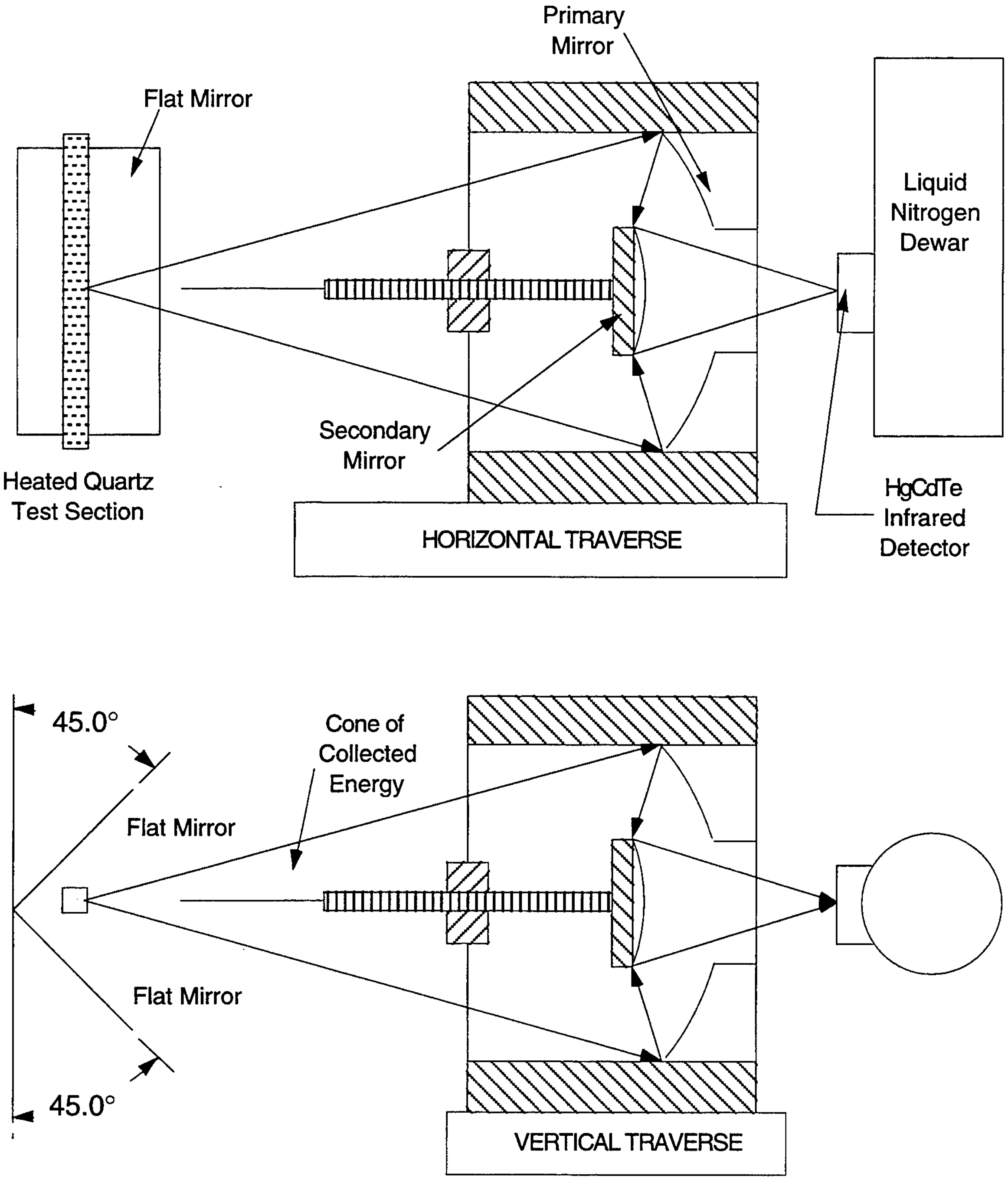


Figure 2.20: Schematic of Infrared Imaging System for Wall Temperature Measurement

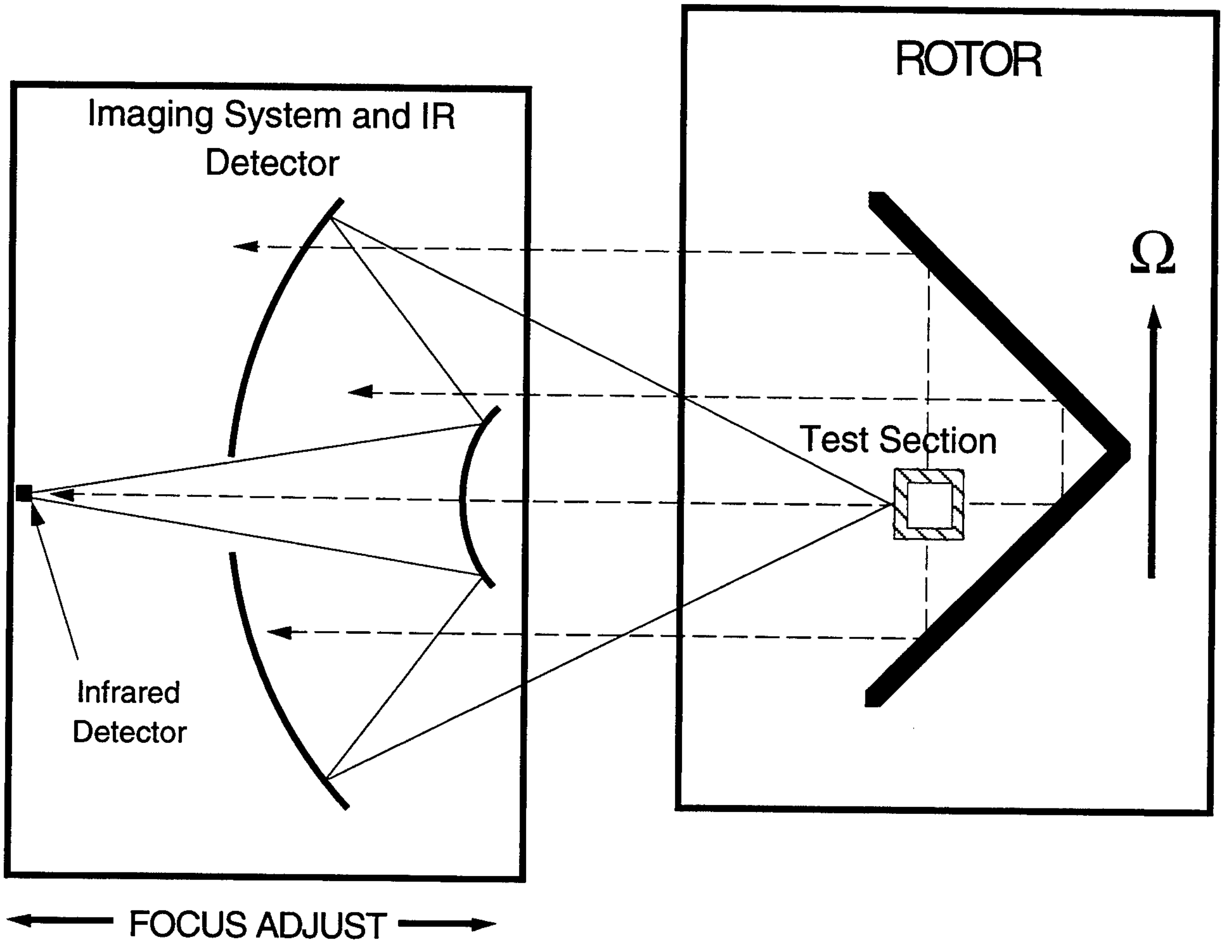
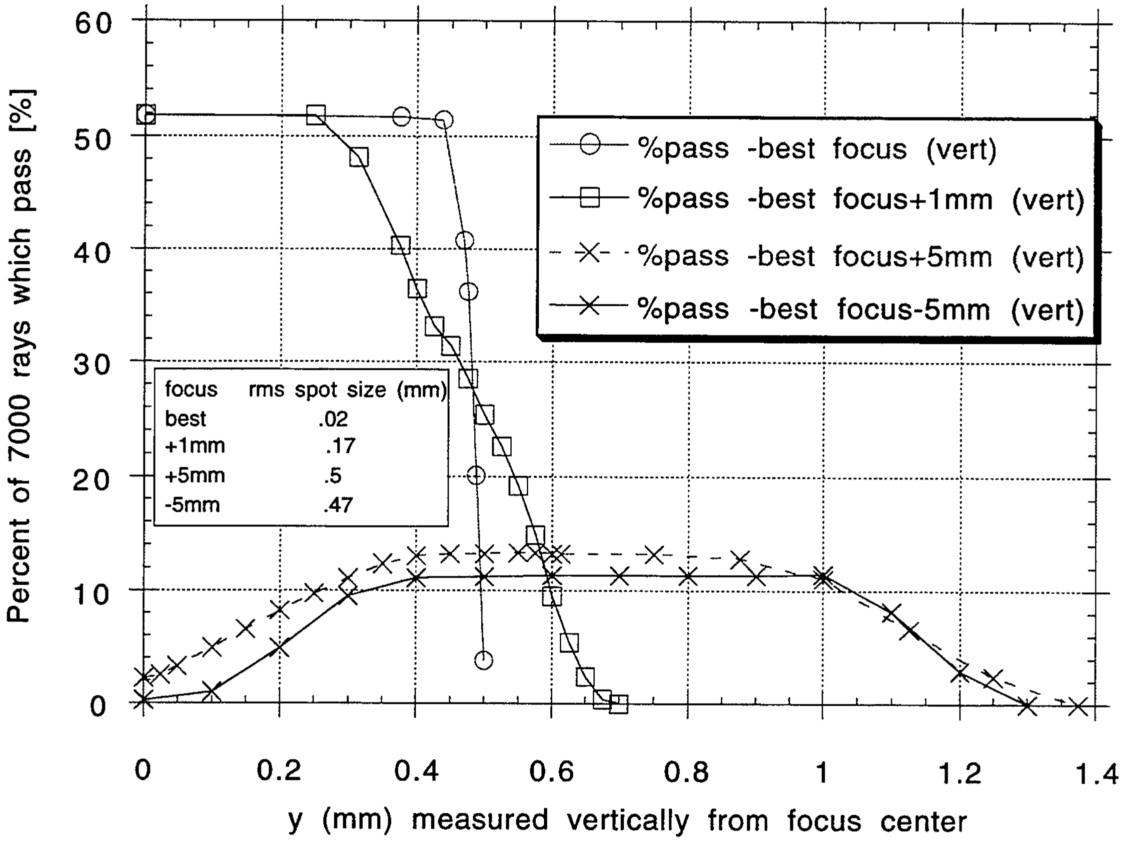


Figure 2.21: Optical Layout for Infrared Measurement System

IR Optics for Internal Heat Transfer Facility: % of rays from test section y-position captured by large collection mirror which go on to impact IR sensor. Various focal positions out of best focus.



focus	rms spot size (mm)
best	.02
+1mm	.17
+5mm	.5
-5mm	.47

Figure 2.22: Infrared Optical System Spatial Resolution Degradation with Focal Shift away from Position of Best Focus.

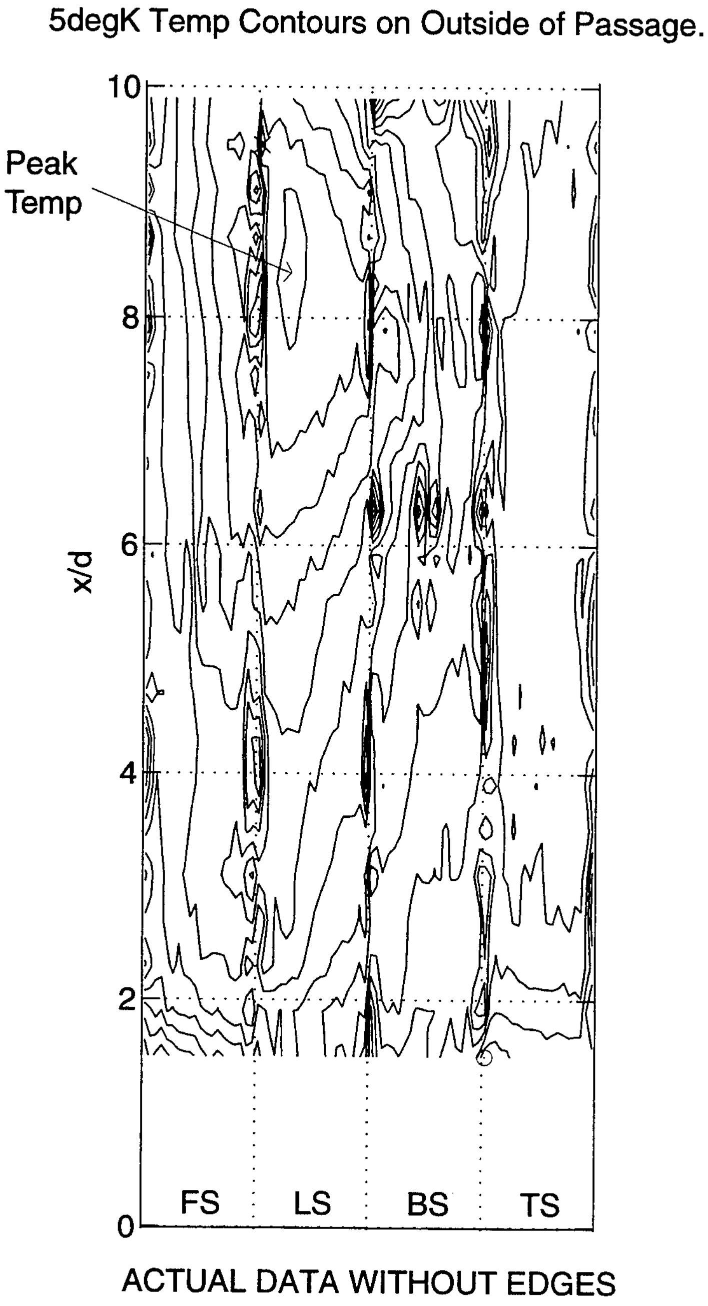


Figure 2.23: Sample Temperature Contours on Passage Exterior from IR Scanner

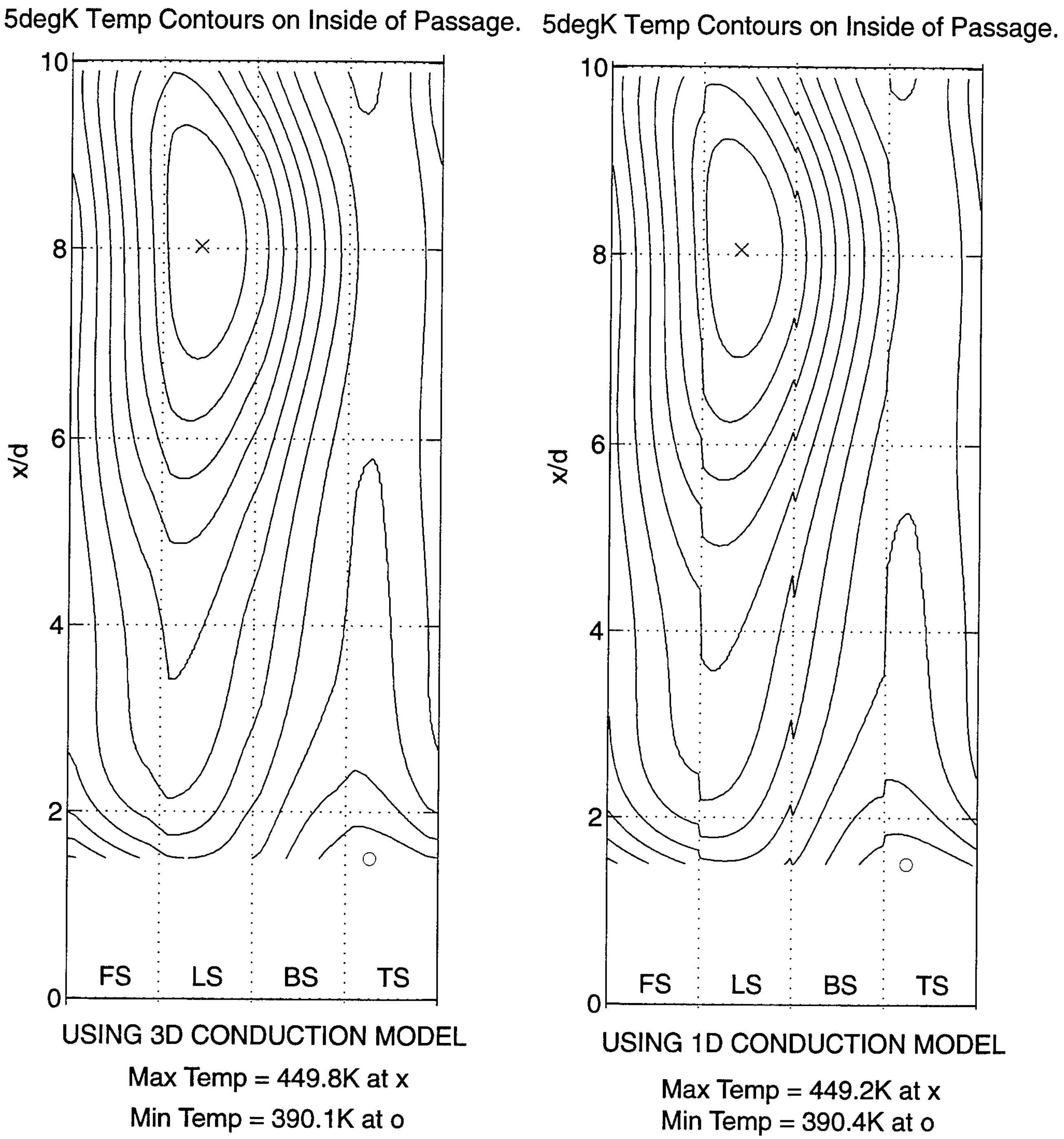


Figure 2.24: Sample Temperature Contours on Passage Interior from 3D & 1D Conduction

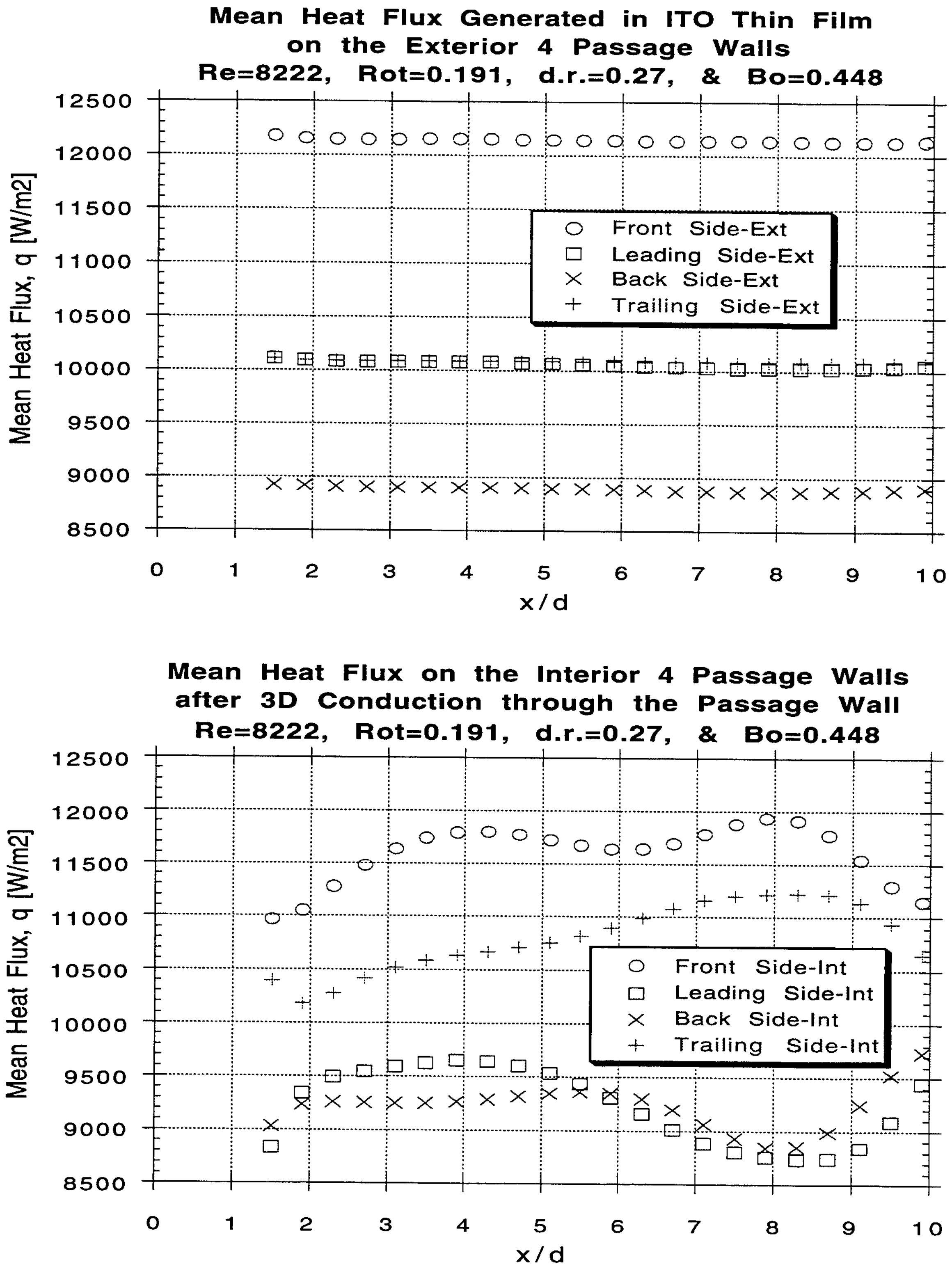
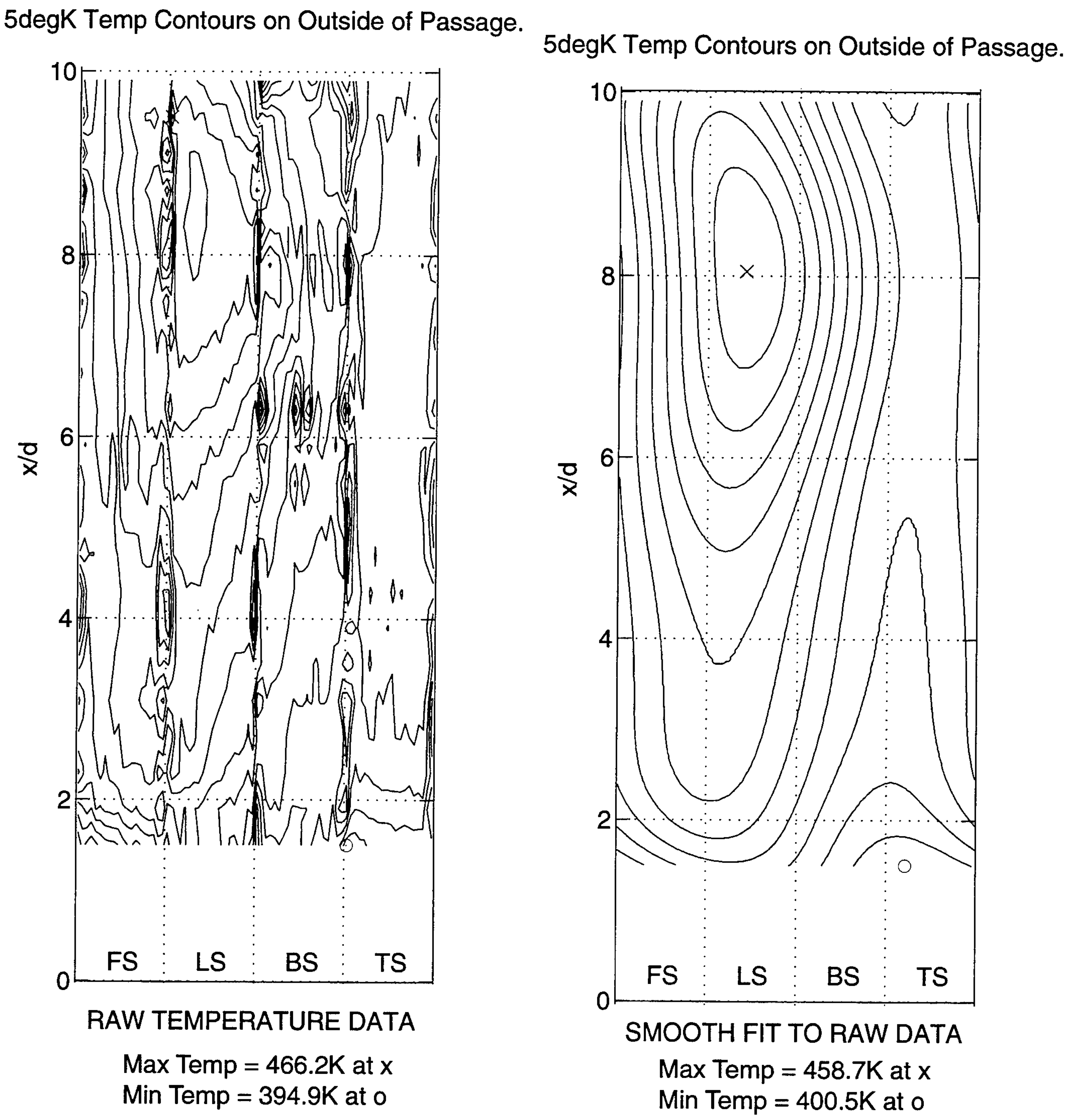


Figure 2.25: Mean Side Heat Flux on External and Internal Passage Surfaces.



RAW TEMPERATURE DATA

Max Temp = 466.2K at x
Min Temp = 394.9K at o

SMOOTH FIT TO RAW DATA

Max Temp = 458.7K at x
Min Temp = 400.5K at o

Figure 2.26: Data and Smooth Fit Temperature Contours on Passage Exterior

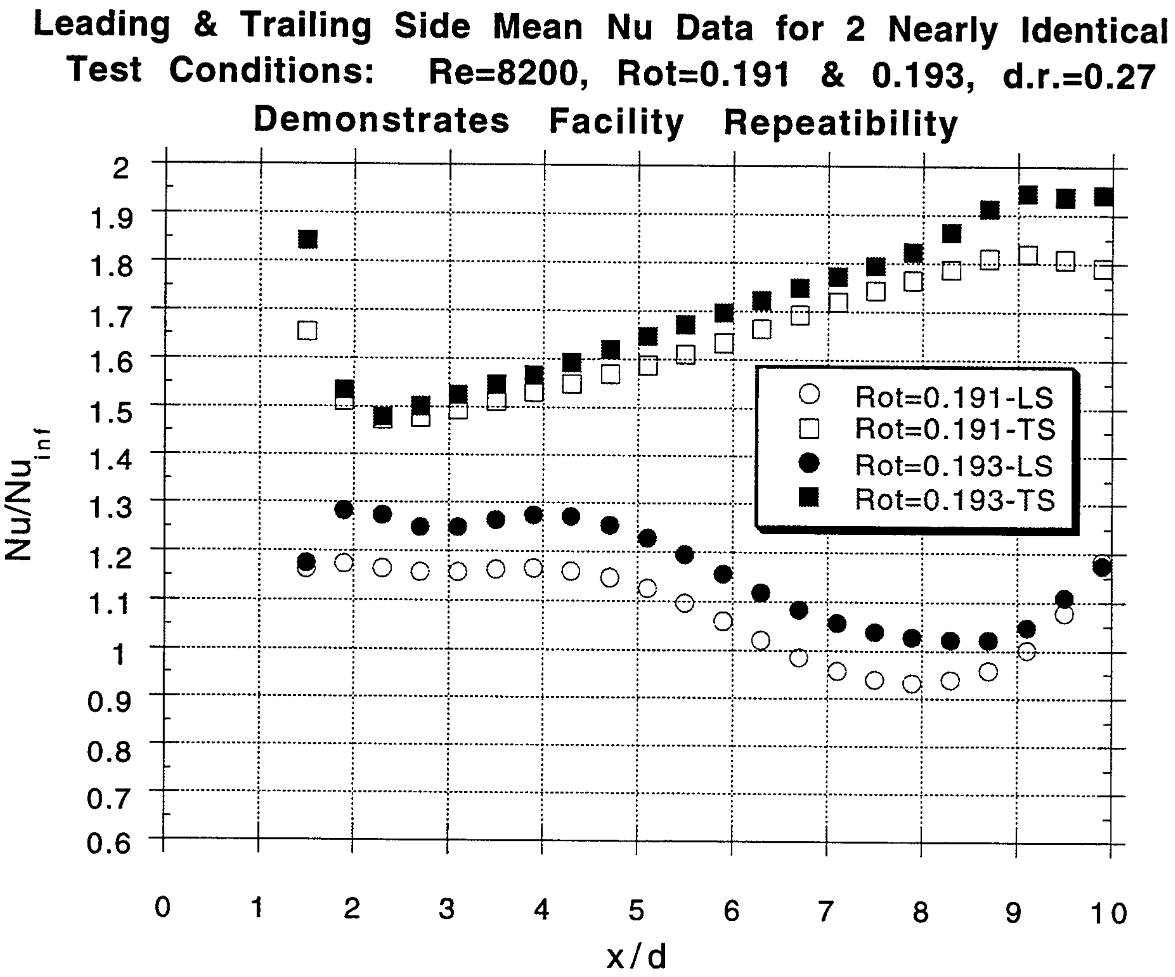


Figure 2.27: Mean Side Nu Data for Leading and Trailing Walls
Data for 2 Nearly Identical Test Cases to Show Repeatability

Chapter 3 - Analytical Model

Introduction

The motivation for constructing a physically based model of the rotating, heated passage flow is two-fold. During the design phase, the model provided an estimate of secondary flow velocities and their spatial extent for assessment of PIV feasibility and image resolution requirements. Then during the subsequent data acquisition and analysis phase, the model allowed for a better understanding of velocity trends and sources of secondary flows. The model is not proposed as a predictive tool for generic rotating flowfields. Several refinements which were added to the model after reviewing the experimental data are also included in this outline. These are declared as empirical adjustments or refinements when they are presented in the text.

To clearly expose the relative importance of various effects, the model is presented in steps of increasing sophistication. This begins with the standard turbulent pipe flow, and then incorporates Coriolis effects and finally buoyancy.

Developing Passage Flow

The approach undertaken here is to assume that the passage flow can be dissected into independent shear flows on each of the four passage walls and a central inviscid "core" region. This methodology has been used with success by other researchers in analyzing rotating internal flows. Mori and Nakayama [40] and Ito and Nanbu [25] pioneered this method for predicting turbine cooling in the 1960's and 1970's. Although there has been much emphasis on 3-D Navier-Stokes solvers, Chew [10] and Govatzidakis [15] have recently found momentum-integral models to be both versatile and useful in predicting the heat transfer in rotating passages.

The validity of this shear layer/core layer approach is substantiated with the following reasoning. Turbine blade cooling passages (and this experimental model in particular) are typically of short length-to-diameter ratio. The l/d ratio is 11.5 for this experiment and ranges from 10 to 25 for aircraft and land-based engines. Also Re_d = 10000 for this experiment, which is usually considered to be transitional or turbulent. The hydrodynamic entry length for turbulent pipe flow is 10 to 15 diameters [38]. So, it seems appropriate to consider the flow in turbine cooling passages with l/d < 15 as developing turbulent flow. In fact, a conservative estimate of the boundary layer thickness on one passage wall using a power law velocity profile is approximately 0.4 d (<0.5 d) at x/d = 7.

In a developing turbulent pipe flow, the boundary layers on each of the four contiguous passage walls grow almost independently with distance from the inlet. Apart from interference at the corners of the square passage, without rotation the only mutual influence the boundary layers have is to jointly close-off the "core" flow. This accelerates the fluid in the center of the passage, creating a higher boundary layer edge velocity.

Once the four wall flows have been separated, their boundary layers can be satisfactorily analyzed using the momentum integral equation [30], derived by applying conservation of mass and momentum to a constant density 2-D boundary layer.

$$\frac{\tau_o}{\rho_o u_\infty^2} + \frac{\rho_o v_o}{\rho_\infty u_\infty} = \frac{d\theta}{dx} + \theta\left[\left(2 + \frac{\delta_d}{\theta}\right)\frac{1}{u_\infty}\frac{du_\infty}{dx}\right] \qquad \text{(Eqn 3.1)}$$

To compute boundary layer growth with distance down the passage (x), a wall shear stress relation must be assumed. As a compromise between accuracy and ease of algebraic manipulation, a power law form of this relation is used in this model.

$$u^+ = 8.75\left(y^+\right)^{1/n} \qquad \text{(Eqn 3.2)}$$

Generally, for Re_x < 1,000,000 an exponent of n=7 provides a "good" match with experimental data for external boundary layers [30,44]. For two significant reasons, holding "n" to a constant value of 7 was not considered adequate for this model. First, there is a substantial body of research [5,19] which indicates that elevated levels of freestream (or "core" in this case) turbulence can reduce the turbulent boundary layer shape factor by creating a higher momentum "fuller" boundary layer. Specifically, freestream turbulence levels of 5-10% are reported to reduce the shape factor (H) by a comparable percentage. For the assumed power law boundary layer profile employed in this model, an exponent value of n=9 yields a shape factor which is 5% less than that for n=7. Since the spatial turbulence levels measured using PIV are in the 5-15% range (ref. chapter 2), a higher exponent of n=9 was deemed appropriate for this model (and indeed provides a better match with experimental data).

As a further refinement based on comparison with velocity data, the power law exponent is allowed to vary when rotation is added to the integral model. With rotation, the boundary layer growth rates are altered by the convection of momentum deficit from wall to wall by secondary crossflows. The model accounts for this transport between boundary layers by adjusting the boundary layer thickness and shape on all four passage walls. How this exchange is accounted for is addressed later in more detail.

The implementation of the momentum integral equation employed in this model follows the steps outlined below and derived in detail in a number of fluid dynamics texts [30,38,44]. Solving Equation 3.2 for the wall shear (τ_o) yields:

$$\tau_o = \left[\frac{\left(\frac{n}{n+1} - \frac{n}{n+2} \right)^{1/n}}{8.75} \right]^{2n/n+1} \rho_\infty u_\infty^2 \left[\frac{\nu}{\theta u_\infty} \right]^{2/n+1} = C^{2n/n+1} \rho_\infty u_\infty^2 \left[\frac{\nu}{\theta u_\infty} \right]^{2/n+1}$$

When this expression is substituted into the integral momentum equation and terms are collected, the following differential equation results.

$$d\left[\theta^{n+3/n+1} u_\infty^{(n+3)(3n+2)/n(n+1)}\right] = \left(\frac{n+3}{n+1}\right) C^{2n/(n+1)} \nu^{2/n+1} u_\infty^{[((n+3)(3+2/n)/n+1)-(2/n+1)]} dx$$

Integrating this expression from x=0 to a point x>0 in the flow passage, with the initial condition that the momentum thickness (θ) is 0 at x=0, produces an expression for $\theta(x)$.

$$\theta(x) = f(u_\infty, n, \nu) \left\{ \int_0^x u_\infty^{g(n)} dx \right\}^{n+1/n+3} \qquad \text{(Eqn 3.3)}$$

where,

$$f(u_\infty, n, \nu) = \frac{\left(\frac{n+3}{n+1}\right)^{(n+1)/(n+3)} C^{2n/(n+3)} \nu^{2/(n+3)}}{u_\infty^{(3n+2)/n}} \qquad \text{and} \qquad g(n) = \frac{(n+3)(3n+2) - 2n}{n(n+1)}$$

To evaluate this relation following the flow each Δx step down the cooling passage, inlet conditions for velocity and temperature are required. Over the years, researchers have noted the significant effect that passage inlet conditions have on rotating heat transfer predictions, especially in the first outflow passage of a multipass calculation. Since experiments are generally conducted with long unheated starting lengths, most codes use a fully developed inlet condition for velocity (Poiseuille flow) and a constant temperature profile. Prakash and Zerkle [45] were first to articulate the need for a fully developed inlet condition for velocity which also accounts for rotation (before the heated section begins). Considering the tapered inlet orifice (Fig 2.3) and short unheated starting length (1.25d) of this experimental facility, constant inlet velocity and temperature profiles were assumed. Then with each Δx step down the passage, the momentum thickness is calculated by approximating Eqn 3.3 as a summation.

$$\theta^m = f(u_\infty, n, \nu)\left\{\sum_{j=1}^{m}\left(u_\infty^j\right)^{g(n)} \Delta x\right\}^{n+1/_{n+3}} \qquad \text{(Eqn 3.4)}$$

At the conclusion of each step, the effect of boundary layer growth on the core flow is assessed by applying conservation of mass to the passage cross-section.

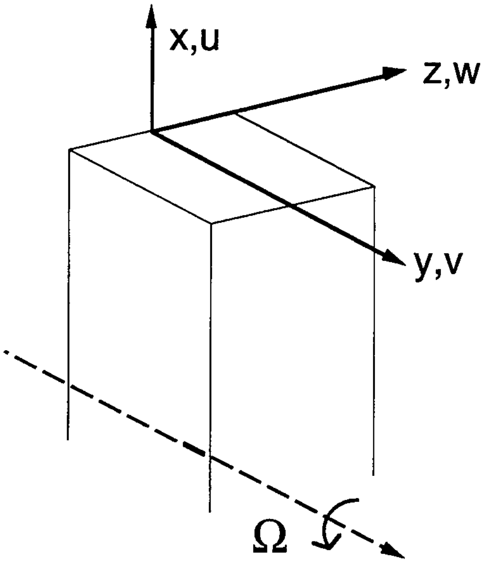
$$\dot{m} = \mathrm{cons\,tan\,t} = \rho_{in} u_{in} d^2 = \int_0^d \int_0^d \rho u dz dy$$

So, the core flow velocity, $u_\infty \equiv u_{in}$, rises as the blockage due to the wall layers increases down the passage. Core acceleration and boundary layer growth predictions using this method are shown in Figures 3.1 and 3.2 for a Reynolds number of 10000. For the case shown, the core acceleration is 12% by x/d of 7 and the boundary layer thickness is 0.34. This is 15% less than the value of 0.4 quoted earlier which didn't account for core acceleration. Since u_∞ is part of the integrand in Eqn 3.3, an accelerating u_∞ energizes the boundary layers and retards their growth relative to a constant freestream velocity case.

Passage Flow with Rotation

The next step is to assess the effect of rotation on the growing boundary layers and the core flow. To do this, the averaged Navier-Stokes equations for steady, incompressible flow are evaluated in a coordinate system fixed to the passage as shown.

$$\left(\bar{u}\cdot\bar{\nabla}\right)\bar{u}+\frac{\partial\left(\overline{u_i'u_j'}\right)}{\partial x_j}=\nu\nabla^2\bar{u}-\frac{\nabla p}{\rho}-2\bar{\Omega}\times\bar{u}$$



and $\bar{\nabla}\cdot\bar{u}=0$

where $\bar{u}=u\hat{i}+v\hat{j}+w\hat{k}$

and $\bar{\Omega}=\Omega\hat{j}$

The $-2\bar{\Omega}\times\bar{u}$ term is the Coriolis acceleration which accounts for the use of a non-inertial reference frame attached to the rotating passage. There are two principle effects of this Coriolis term which are evident in the component form of the Navier Stokes equations.

x-momentum:

$$u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}+\frac{\partial\overline{u'^2}}{\partial x}+\frac{\partial\overline{u'v'}}{\partial y}+\frac{\partial\overline{u'w'}}{\partial z}=\nu\nabla^2u-\frac{1}{\rho}\frac{\partial p}{\partial x}-2\Omega w$$

y-momentum:

$$u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}+\frac{\partial\overline{v'u'}}{\partial x}+\frac{\partial\overline{v'^2}}{\partial y}+\frac{\partial\overline{v'w'}}{\partial z}=\nu\nabla^2v-\frac{1}{\rho}\frac{\partial p}{\partial y}$$

z-momentum:

$$u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}+\frac{\partial\overline{w'u'}}{\partial x}+\frac{\partial\overline{w'v'}}{\partial y}+\frac{\partial\overline{w'^2}}{\partial z}=\nu\nabla^2w-\frac{1}{\rho}\frac{\partial p}{\partial z}+2\Omega u$$

The +2Ωu term in the z-momentum equation creates the well-documented Coriolis double vortex associated with pipe flow rotating in the orthogonal mode. Then, as the secondary flow due to the +2Ωu term develops, the Coriolis effect associated with the -2Ωw term in the x momentum equation becomes important. This -2Ωw term provides additional streamwise acceleration to flow near the wall which is flowing in the -z direction (from trailing wall to leading wall) due to the Coriolis vortex. These two distinct effects are incorporated into the model in the following manner.

First Coriolis Component

Evaluating the z-momentum equation in the "core" region of the passage, where the cross-stream velocity, w, is small and the inertial and viscous terms can be neglected, yields the following:

$$\frac{dp}{dz} = 2\Omega u_{\infty}\rho_{\infty}$$

This is the form of the transverse pressure gradient produced by rotation. Substituting this expression for dp/dz into the same z-momentum equation now evaluated in the side-wall boundary layer ($y \approx 0$ or d) produces:

$$\rho u \frac{dw}{dx} = -2\Omega u_{\infty}\rho_{\infty} + 2\Omega u\rho$$

While the viscous and turbulent inertial terms obviously play important roles in the near wall region, they are absent from this formulation. This is done to allow a simple linear expression for the Coriolis crossflow velocity. The effects of viscosity and turbulent inertia are accounted for in the use of the 1/9th power law streamwise velocity profile and in the no-slip condition imposed subsequently on the secondary crossflow.

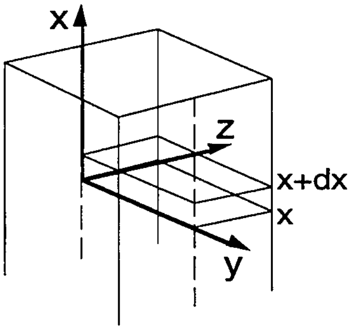
For purposes of the model, this relation is discretized to form the following expression for Δw_{cor} (the added Coriolis-induced flow component for a Δx step down the passage).

$$\Delta w_{cor} \cong 2\Omega\left[\frac{u\rho}{u_{\infty}\rho_{\infty}} - 1\right]\Delta x \qquad \text{(Eqn 3.5)}$$

Any region where uρ < u_∞ ρ_∞ will generate $\Delta w_{cor} < 0$. This condition is satisfied in the boundary layer on both sides of the passage since the velocity at the side walls must ultimately equal zero. In this way, the effect is symmetric about the passage centerline (y/d = 0.5). Flow symmetry about this plane is assumed for all the discussion that follows. With each subsequent step, Δw_{cor} is accumulated to create a secondary crossflow, w_{cor} .

This secondary Coriolis-induced flow near the side-wall must satisfy conservation of mass and no-slip condition at the wall. Applying conservation of mass to a half-passage control volume as shown:

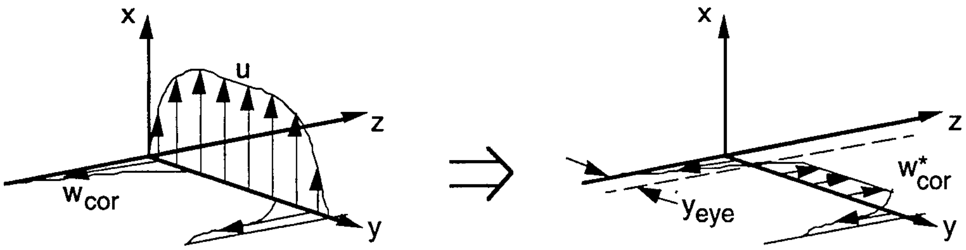
$$\int_0^d \int_x^{x+\Delta x} \rho w_{cor} dxdy - \int_0^d \int_0^{d/2} \rho u_{x+\Delta x} dzdy + \int_0^d \int_0^{d/2} \rho u_x dzdy = 0$$



If the streamwise acceleration is small, the second two terms are very nearly equal and can be ignored. This leaves only the first term in the conservation integral. Then if ρw_{cor} is assumed to vary slowly with x, the first term can be approximated as a line integral in y.

$$\int_0^d \rho w_{cor} dy = 0$$

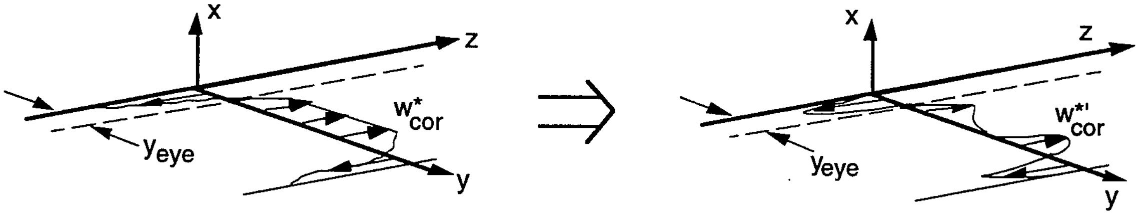
To satisfy secondary flow continuity, a mean value of ρw_{cor} is computed and deducted from the w_{cor} distribution computed in equation 3.5 . w_{cor}* is the adjusted profile which now satisfies continuity.



$$\left(w_{cor}\rho\right)^* = w_{cor}\rho - \frac{\int_0^{d/2} \rho w_{cor} dy}{d/2}$$

By observation, the $w_{cor}*$ profile is indicative of a vortex in each half-passage. Also, the vortex eye (y_{eye}), where $w_{cor}* = 0$, is not centered on the half-passage but is near the side wall. The location of the vortex center is dependent on the side-wall boundary layer thickness.

With secondary flow continuity satisfied, the massflow in the $w_{cor}* < 0$ region must now satisfy the no-slip wall condition. The model employs a nearly parabolic velocity profile here while preserving the total massflow in the negative z direction. The flow profile in the region $w_{cor}* > 0$ is based on two additional physical constraints: continuity of shear (dw/dy) at $y = y_{eye}$ and continuity of shear at y=d/2 due to symmetry. The final secondary velocity profile is indicated below:



This Coriolis-induced secondary flow continues to grow with distance down the passage. Its effect is not limited to the boundary layer along the side-wall since the vortex must negotiate the corners of the passage. Accordingly, flow profiles which preserve the massflow in the vortex are used on both the leading and trailing walls, creating a secondary flow picture in the y-z plane similar to Figure 3.3.

Since the Coriolis secondary flow is confined primarily to the boundary layer region of the passage, the vortex moves low momentum boundary layer fluid from the trailing wall to the side wall, and from there to the leading wall. The leading wall boundary layer then becomes the repository for the momentum deficit convected from the other walls. This contributes to an enhanced thickening of the boundary layer there. Likewise the vortex transports high momentum core fluid into the trailing wall boundary layer. This energizes the boundary layer here, reducing its thickness. To account for these effects, the magnitude of convected momentum deficit is assessed at each step. The three boundary layer displacement thicknesses are then adjusted up or down depending on the sign of the convection (up for the leading wall boundary layer as it thickens and down for the trailing wall boundary layer as it becomes thinner). For example, the additional momentum deficit for the leading wall is:

$$\Delta\delta_{dLS} = \int_{0}^{yeye}\left[\left(1-\frac{u_{BS}\rho_{BS}}{u_{\infty}\rho_{\infty}}\right)w_{cor}\Delta t\right]dy$$

The adjustment to the leading wall boundary layer displacement thickness (δ_{dLS}) is then:

$$\delta_{dLS}^{m'} = \delta_{dLS}^{m} + \frac{\Delta\delta_{dLS}}{d/2}$$

where the momentum defect convected from the side wall has been smeared out over the entire half-passage width. Similar formulations are employed for the trailing and side wall boundary layers as well.

In addition to this change in displacement thickness, as mentioned earlier this momentum defect addition or removal alters the shape of the affected boundary layer. The boundary layer shape is determined by the value of the exponent “n” in the assumed power law velocity profile. For the power law profile shape, the displacement thickness can be expressed as a function of boundary layer thickness, δ, and the power law exponent, n.

$$\delta_{d} = \delta\left[1-\frac{n}{n+1}\right] \rightarrow n = \frac{\delta}{\delta_{d}} - 1$$

If a small change in δ_d were assumed to alter only the profile shape (n) and not its thickness (δ), the resultant exponent (n') would be:

$$n' = \frac{\delta_{d}}{(\delta_{d} + \Delta\delta_{d})\left[1-\frac{n}{n+1}\right]} - 1$$

This exponent adjustment was added to the model after reviewing the experimental velocity data. It was found that using a fraction of the $\Delta\delta_d$ to produce a shape adjustment provided a better match for the trailing and leading wall boundary layers. Since an adjustment in the boundary layer shape implies active mixing of the defect into the boundary layer, the fraction of $\Delta\delta_d$ which is leveraged to produce a change in the profile shape ($n \Rightarrow n'$) is calculated based on a ratio of turbulent to inertial timescales in the flow.

$$\text{fraction} = \frac{u'\delta}{u_{\infty}\Delta x}\Delta\delta_{d}$$

This implementation, while only an empirical refinement, produces a physically believable result. The profile exponent drops on the leading wall from 9 at the inlet to approximately 5 at x/d=10 as the boundary layer thickens with rotation. Conversely, the profile exponent on the trailing wall rises from 9 to nearly 14 as the boundary layer thins here. If there is no rotation, no adjustment occurs.

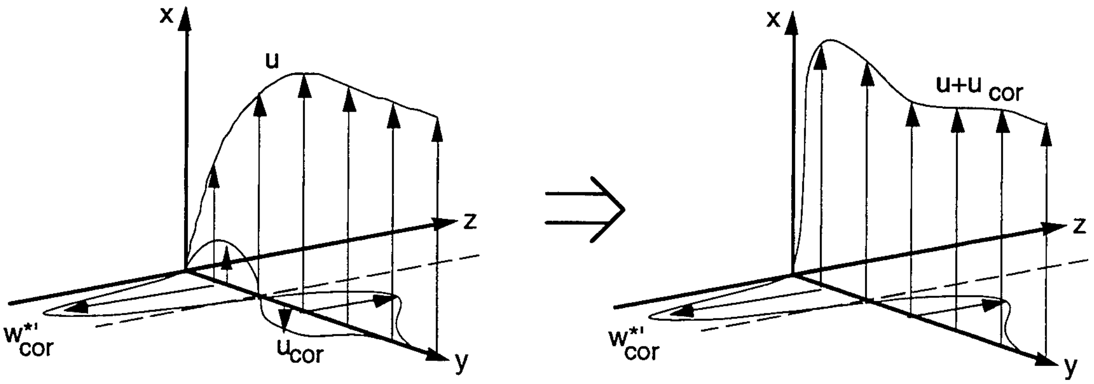
To insure that this modified boundary layer information (both exponent and displacement thickness) is carried over to the next step in the model an "equivalent x" is calculated for each boundary layer at the end of each step. This "equivalent x" is computed using the same boundary layer equation (Eqn. 3.4 solved for x) and represents the x-location corresponding to the new core velocity and displacement thickness after the above adjustments have been made. Figures 3.4 and 3.5 show u(y) and w(y) calculations for the nominal condition of Re=10000 and Rot=0.2 compared to Rot=0.0.

Second Coriolis Component

As mentioned previously, the Coriolis acceleration has a second component which effects the streamwise (x) momentum equation. Following the same methodology used for the z-momentum component, the increment of streamwise acceleration (u du/dx) attributed to the -2Ωw term is assessed as:

$$\Delta u_{cor} \cong -2\Omega \frac{w\rho}{u_{\infty}\rho_{\infty}} \Delta x \qquad \text{(Eqn 3.6)}$$

From the above relation, in the region near the side wall (where w<0), this will result in an acceleration, whereas in the core region (where w>0) there will be a corresponding deceleration.



These two effects are incorporated into the model as follows. First, at each step the additional component of Coriolis-induced streamwise velocity, Δu_{cor}, is computed using equation 3.6. Based on comparisons with data (see chapter 4) the added velocity is allowed to grow independent of the power law streamwise profile. The sum of Δu_{cor} from each step is superposed on the streamwise velocity profile at the end of each step, creating a local bulge in the side wall boundary layer shape. To match the experimental data, this u_{cor} flow is diffused away from the wall by a distance related to the turbulence level in the flow:

$$\Delta y_{diff} \propto \frac{u'}{u_{\infty}} \Delta x$$

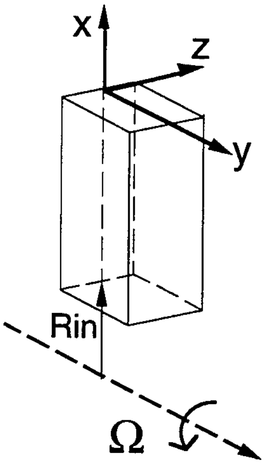
The second effect of the streamwise Coriolis acceleration impacts the passage core. Here Δu_{cor} is negative and thus decelerates the core velocity pushing massflow toward the boundary layers and thus squeezing the boundary layers back toward the walls. To incorporate this in the model, all of the boundary layer displacement thicknesses are reduced by an amount equal in sum to the net mass defect associated with $\Delta u_{cor} < 0$. This has the desired effect of thinning the boundary layers and decelerating the core.

The combined effect of these two additions to the model are shown in Figures 3.6 and 3.7 which compare the model predictions with and without this component. The result of incorporating the streamwise Coriolis effect in this manner is a thinner side wall boundary layer and a reduced core acceleration (Fig 3.6). As the side wall boundary layer decreases, so does the distance from the vortex eye ($w_{cor} = 0$) to the wall (Fig 3.7). This squeezing of the vortex closer to the wall in turn increases the local secondary flow velocities here. This effect is not linear with rotational frequency, as can be seen from Eqn 3.6. Δu_{cor} is proportional to Ωw , and since w is proportional to Ω , Δu_{cor} varies with Ω^2. This dependence is discussed again in chapter 4.

Buoyancy due to Heat Addition

To this point the entire discussion has been isothermal (and constant density). To account for the effect of temperature gradients in the flow, we must revisit the governing Navier-Stokes equations.

$$(\vec{u}\cdot\vec{\nabla})\vec{u}+\frac{\partial\left(\overline{u_i' u_j'}\right)}{\partial x_j}=\nu\nabla^2\vec{u}-\frac{\nabla p}{\rho}-2\vec{\Omega}\times\vec{u}$$
$$+\beta(T-T_\infty)\left(\vec{\Omega}\times\vec{\Omega}\times\vec{R}\right)$$



where $\vec{u}=u\hat{i}+v\hat{j}+w\hat{k}$

$\vec{R}=(R_{in}+x)\hat{i}+y\hat{j}+z\hat{k}$

and $\vec{\Omega}=\Omega\hat{j}$

Here the Boussinesq [7] approximation has been applied, allowing us to consider the effects of variable density only in the buoyancy source term. There are likewise two components of this buoyancy acceleration term which are evident from the component equations.

x-momentum:

$$u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}+\frac{\partial\overline{u'^2}}{\partial x}+\frac{\partial\overline{u'v'}}{\partial y}+\frac{\partial\overline{u'w'}}{\partial z}=$$
$$\nu\nabla^2 u-\frac{1}{\rho}\frac{\partial p}{\partial x}-2\Omega w-\Omega^2(R_{in}+x)\beta(T-T_\infty)$$

y-momentum:

$$u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}+\frac{\partial\overline{v'u'}}{\partial x}+\frac{\partial\overline{v'^2}}{\partial y}+\frac{\partial\overline{v'w'}}{\partial z}=\nu\nabla^2 v-\frac{1}{\rho}\frac{\partial p}{\partial y}$$

z-momentum:

$$u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}+\frac{\partial\overline{w'u'}}{\partial x}+\frac{\partial\overline{w'v'}}{\partial y}+\frac{\partial\overline{w'^2}}{\partial z}=$$
$$\nu\nabla^2 w-\frac{1}{\rho}\frac{\partial p}{\partial z}+2\Omega u-\Omega^2 z\beta(T-T_\infty)$$

The additional term in the z-momentum equation produces a positive acceleration on fluid near the leading wall ($T>T_\infty$ in the boundary layer and $z<0$ for the leading wall). This thickens the boundary layer here. Likewise for the trailing wall (where $z>0$), this acceleration (now of negative sign) accelerates hot trailing wall fluid toward the passage center. Since this buoyancy-induced center seeking fluid motion is perpendicular to the passage wall, it is modeled as mass injection from the wall into the boundary layer with normal velocity, v_o . Returning to the general momentum integral equation with wall injection (Eqn 3.1):

$$\frac{\tau_o}{\rho_o u_\infty^2} + \frac{\rho_o v_o}{\rho_\infty u_\infty} = \frac{d\theta}{dx} + \theta\left[\left(2 + \frac{\delta_d}{\theta}\right)\frac{1}{u_\infty}\frac{du_\infty}{dx}\right]$$

Isolating the effect of v_o on θ in this relation, $v_o > 0$ increases the momentum thickness with x. As an approximation,

$$\Delta\theta \cong \frac{\rho_o v_o}{\rho_\infty u_\infty}\Delta x$$

To incorporate this effect into the model, the mean buoyancy-induced normal velocity is computed for both the leading and trailing wall boundary layers. This mean velocity is used as the characteristic normal velocity (v_o) to compute $\Delta\theta$ for both boundary layers. The adjustment in θ is then assessed similarly to those adjustments created by the Coriolis effect. As expected, the net effect is to thicken both the trailing and leading wall boundary layers as the density ratio > 0. This in turn creates a greater core acceleration.

The second buoyancy term is in the x-momentum equation and actually opposes the mean flow direction. An instructive way to visualize the effect of this acceleration is to consider a long, narrow cooling passage rotating in the orthogonal mode that is capped at both ends. Heat is added to the contained fluid from the walls and because of the centrifugal body force, hot fluid near the walls is accelerated in toward the axis of rotation. Since the fluid is trapped in the passage, in steady state there can be no mean flow. Then by conservation of mass, the colder core fluid is accelerated away from the axis of rotation. Far from the capped passage ends, the flowfield resembles free convection currents on four walls toward the axis of rotation with a bulk core motion in the opposing direction.

The model employs one of two alternative strategies for merging this flowfield with a bulk mean flow. In the first strategy, the buoyant flow computed at each time step is treated as a momentum defect in the boundary layer. High local turbulence mixes out the defect and increases the boundary layer momentum thickness without causing separation or reverse flow. This produces a rapid thickening of the leading wall boundary layer, since most of the hot boundary layer fluid collects here due to the action of the Coriolis vortex. Figure 3.8 shows the predicted boundary layer thickness using this implementation compared to the cases for Rot=0.0 & 0.2 with d.r.=0.0. Evident is a strong core acceleration reaching nearly 20% at x/d=8.

The second strategy of accounting for the buoyant flow allows a separation bubble to occur near the leading wall. The bubble's spatial extent matches the mass defect of the buoyant flow and essentially pushes the turbulent boundary layer away from the physical wall to a point inside the tube. To assess the impact of this alternate strategy where no mixing occurs, the identical case was run with a separation bubble on the leading surface. The other three walls are subject to strong secondary flows and are considerably thinner, so the buoyancy associated with each of them is either convected to the leading wall or mixed into the respective boundary layer momentum thickness. The separation bubble of stagnant fluid creates a strong core acceleration to nearly twice that of the previous method as shown in Figure 3.9. The actual flowfield measurements shown in the next chapter match more closely the case with a separated leading wall boundary layer. The alternate effects of stagnant, separated leading wall flow and boundary layer thickening with buoyancy have been calculated by other researchers [6,13,45,46,47].

To generate the above model predictions, temperature profile shapes must be assumed in order to compute the magnitude of the buoyant acceleration terms. To this end, a turbulent Prandtl number of unity is assumed, constraining the thermal boundary layer thickness to match the hydrodynamic boundary layer thickness. The other essential parameter is the wall temperature(s). Since this model was employed exclusively for data interpretation, experimentally measured wall temperature data and surface heat flux values are used to initiate the model. As such, the model can not be used to provide an independent calculation of Nu for comparison with experiment. Given the thermal boundary layer endpoints ($T=T_\infty$ at $y = \delta_t = \delta$ and $T=T_{wall}$ at $y = 0$), a power law profile is used to estimate the boundary layer shape. The exponent of the profile is determined using energy conservation for each boundary layer (accounting for energy convected into or out of the boundary layer by the secondary flow system).

Final Appraisal

In summary, a momentum integral model has been developed to calculate the flowfield in this rotating, heated passage flow. The model assumes a turbulent flowfield with secondary flows creating mass and momentum exchange between otherwise independently developing boundary layers. Numerous refinements were added to improve comparison with the experimental data to be presented in chapter 4. As will be seen shortly, despite the approximations employed the model is able to capture both qualitative and quantitative trends in the measured data remarkably well. However, the value of the model is as an aid to interpreting the measured velocity field rather than as a stand-alone predictive tool. A copy of the C code used to implement the model is found in Appendix A.

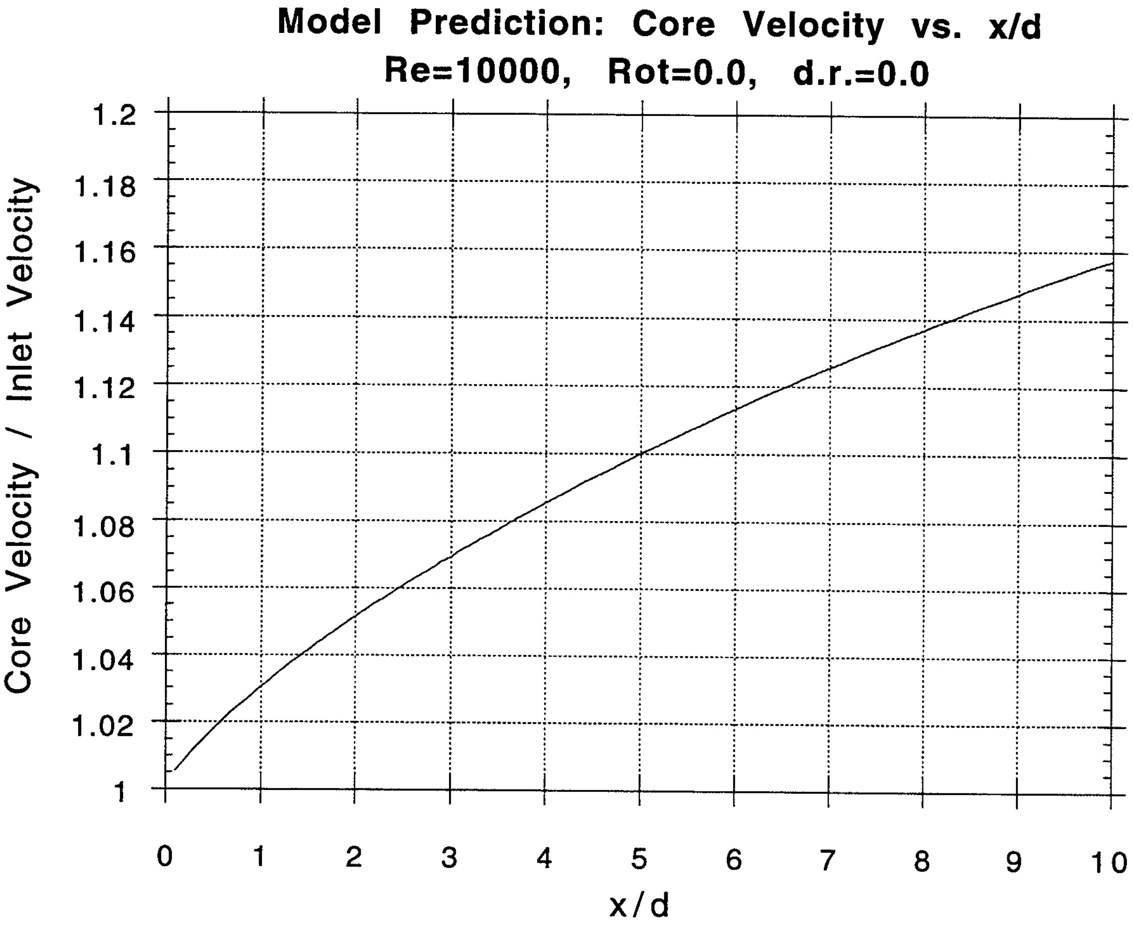


Figure 3.1: Model Prediction of Core Acceleration with Distance from Inlet

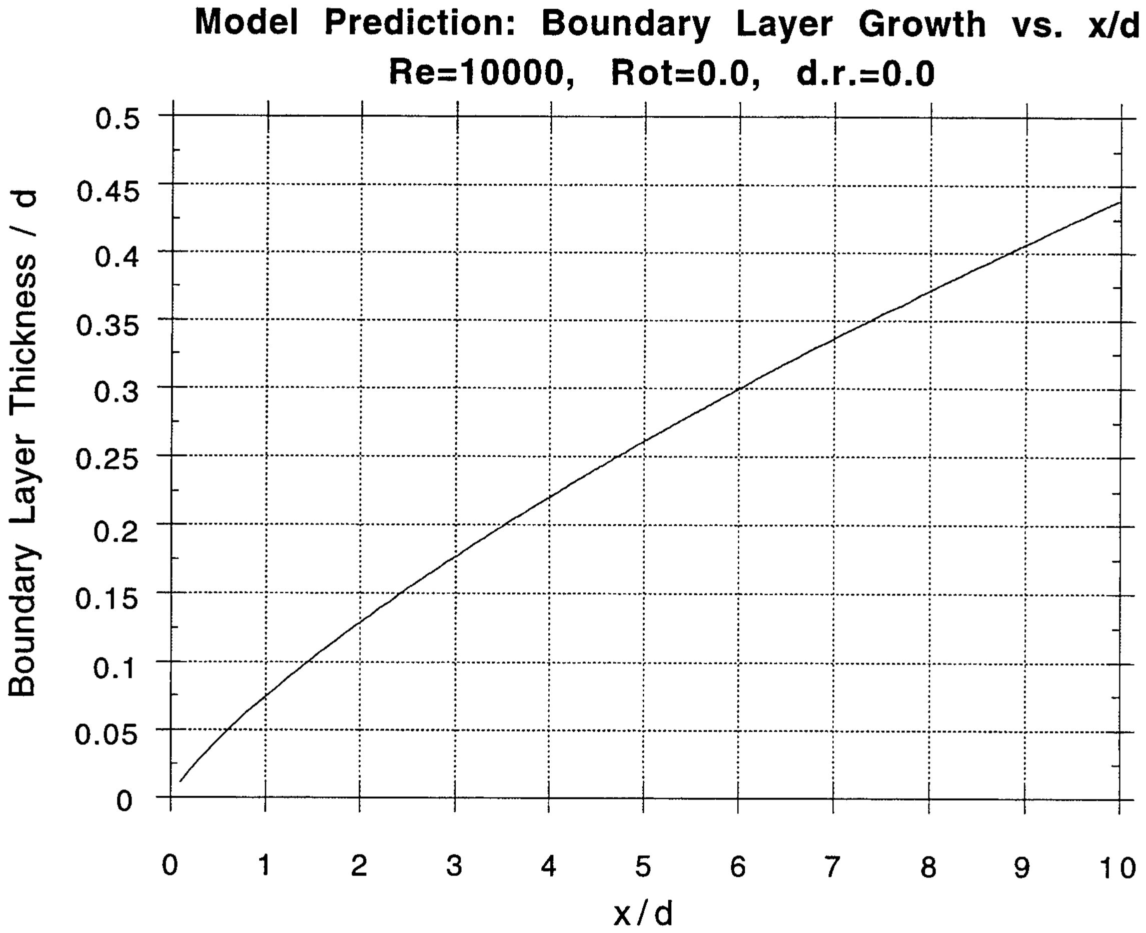


Figure 3.2: Model Prediction of Boundary Layer Growth with Distance from Inlet

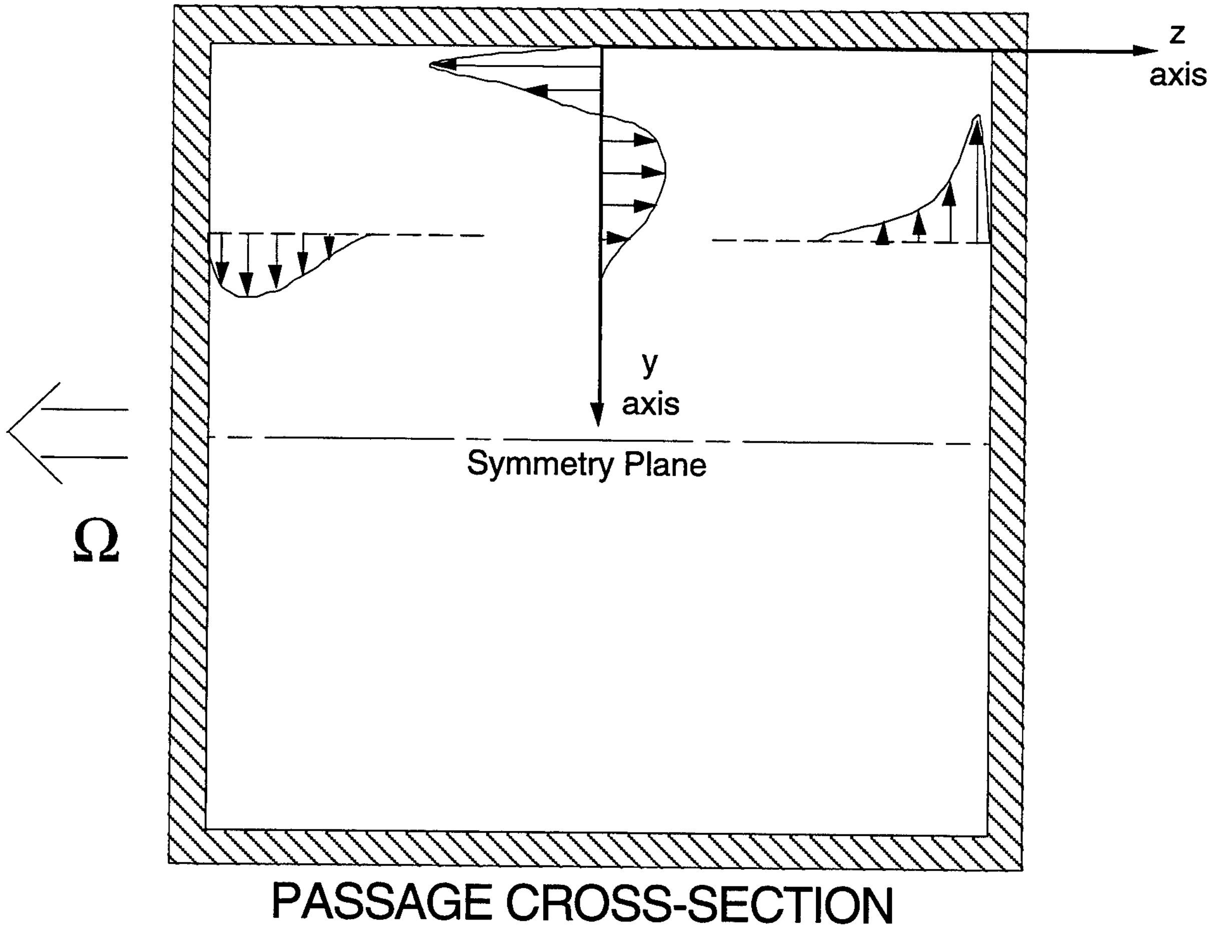


Figure 3.3: Model Schematic of Coriolis Vortex Crossflows

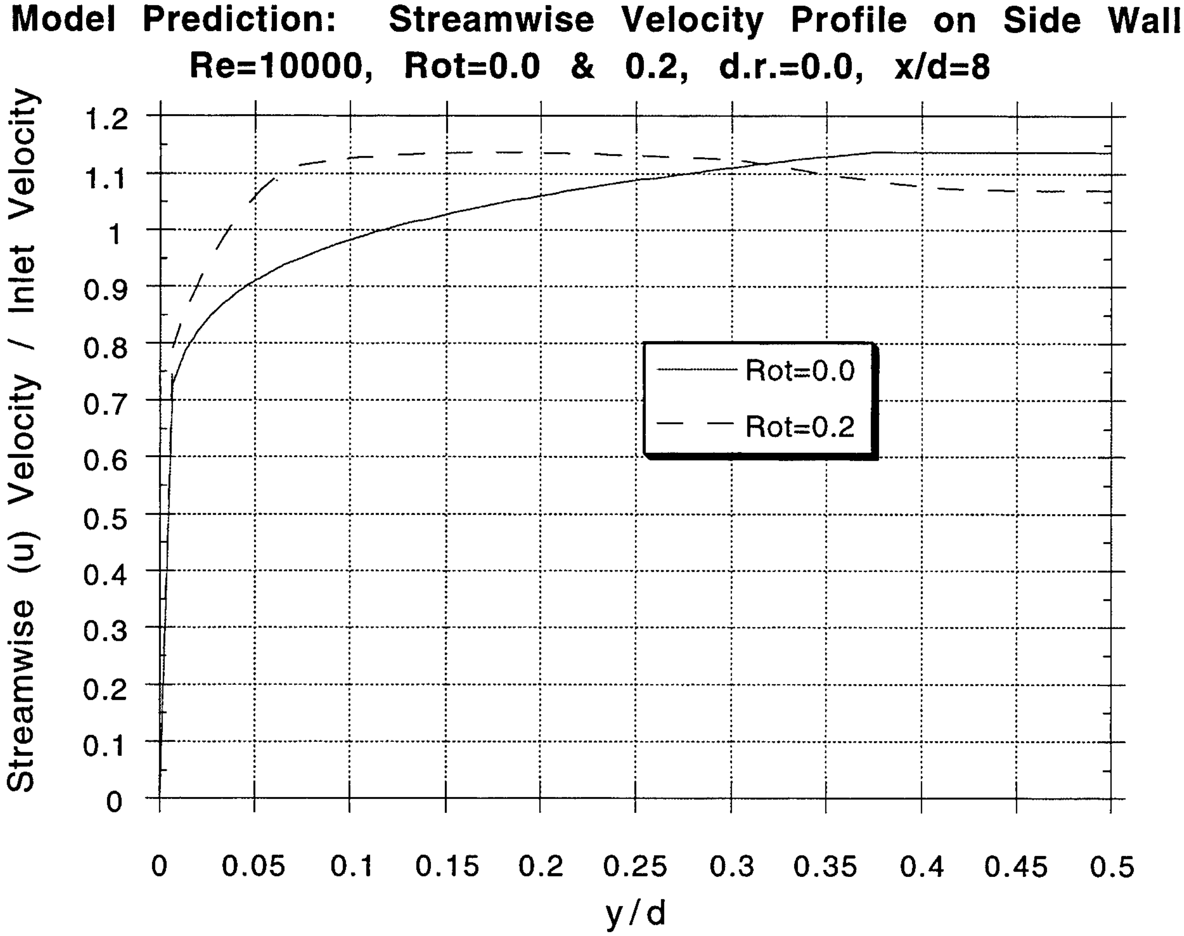


Figure 3.4: Model Prediction of Rotation Effect on Side Wall Boundary Layer Profile

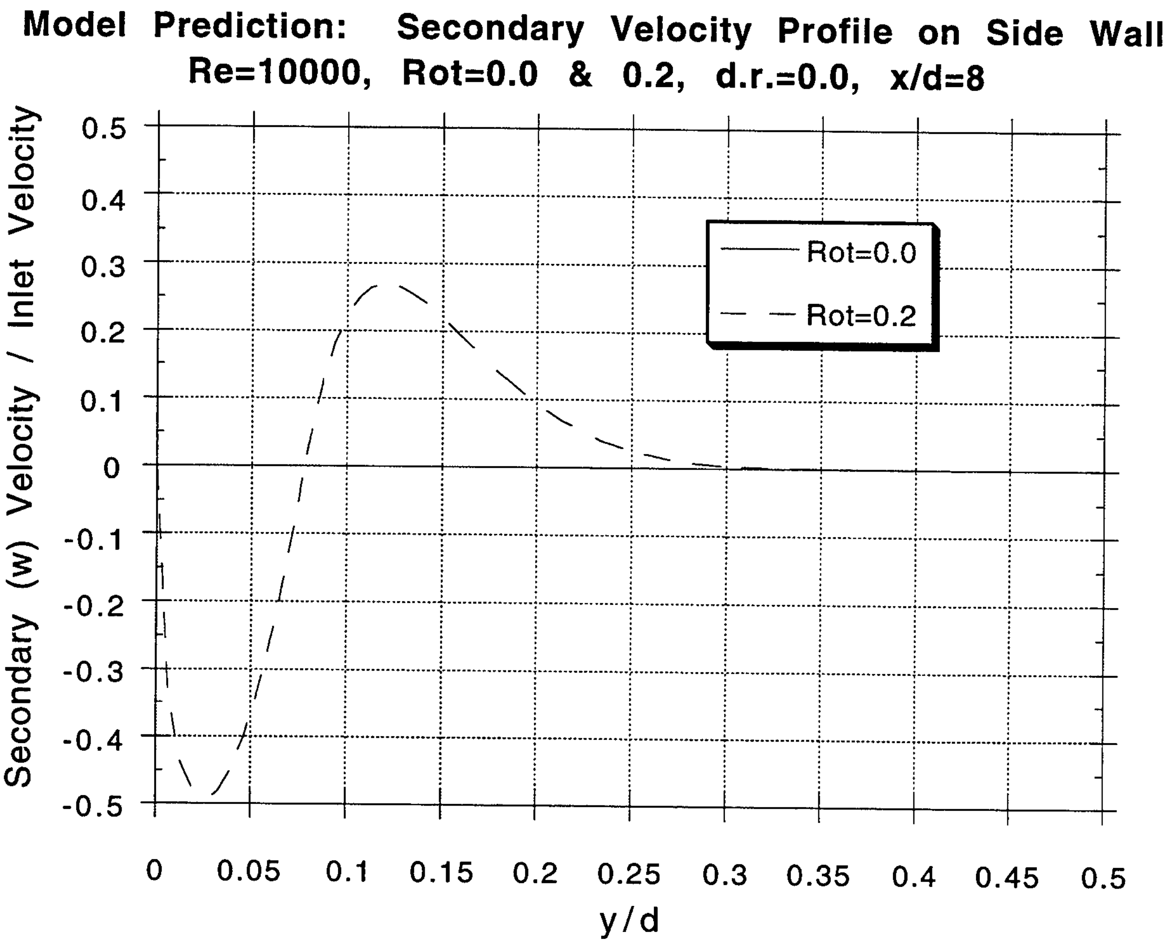


Figure 3.5: Model Prediction of Coriolis Induced Crossflow along Side Wall

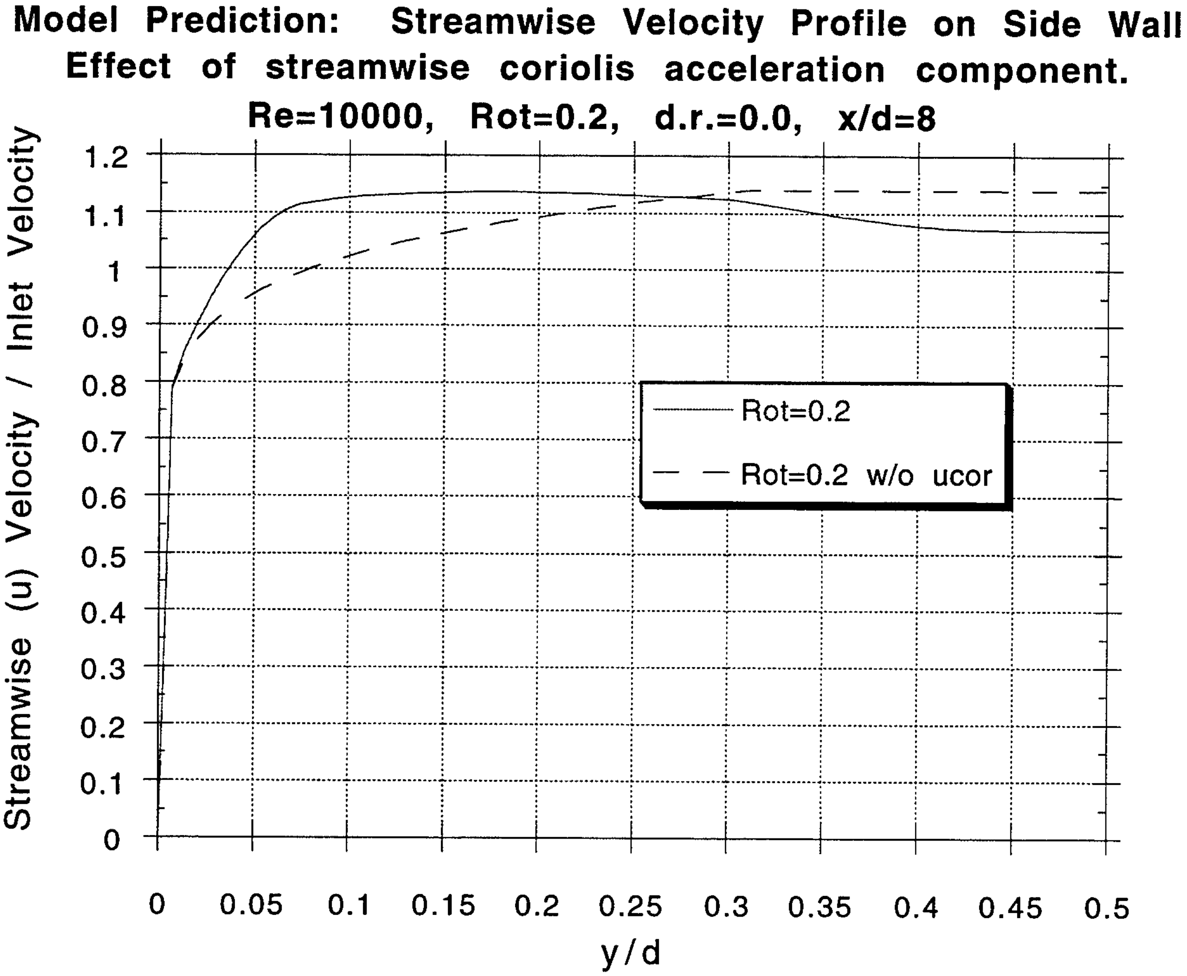


Figure 3.6: Model Prediction of Rotation Effect on Side Wall Boundary Layer Profile
Effect of Streamwise Component of Coriolis Acceleration

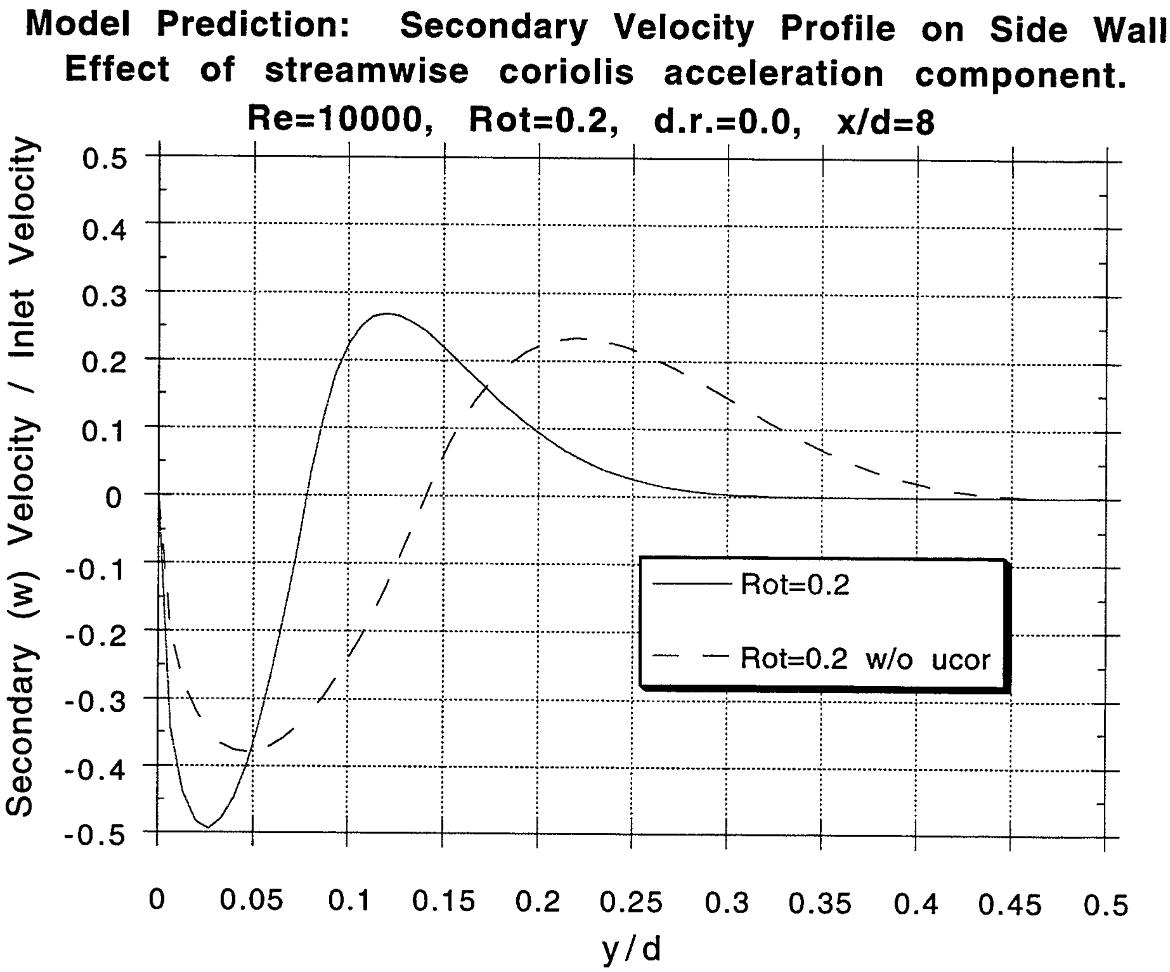


Figure 3.7: Model Prediction of Coriolis Induced Crossflow along Side Wall
Effect of Streamwise Component of Coriolis Acceleration

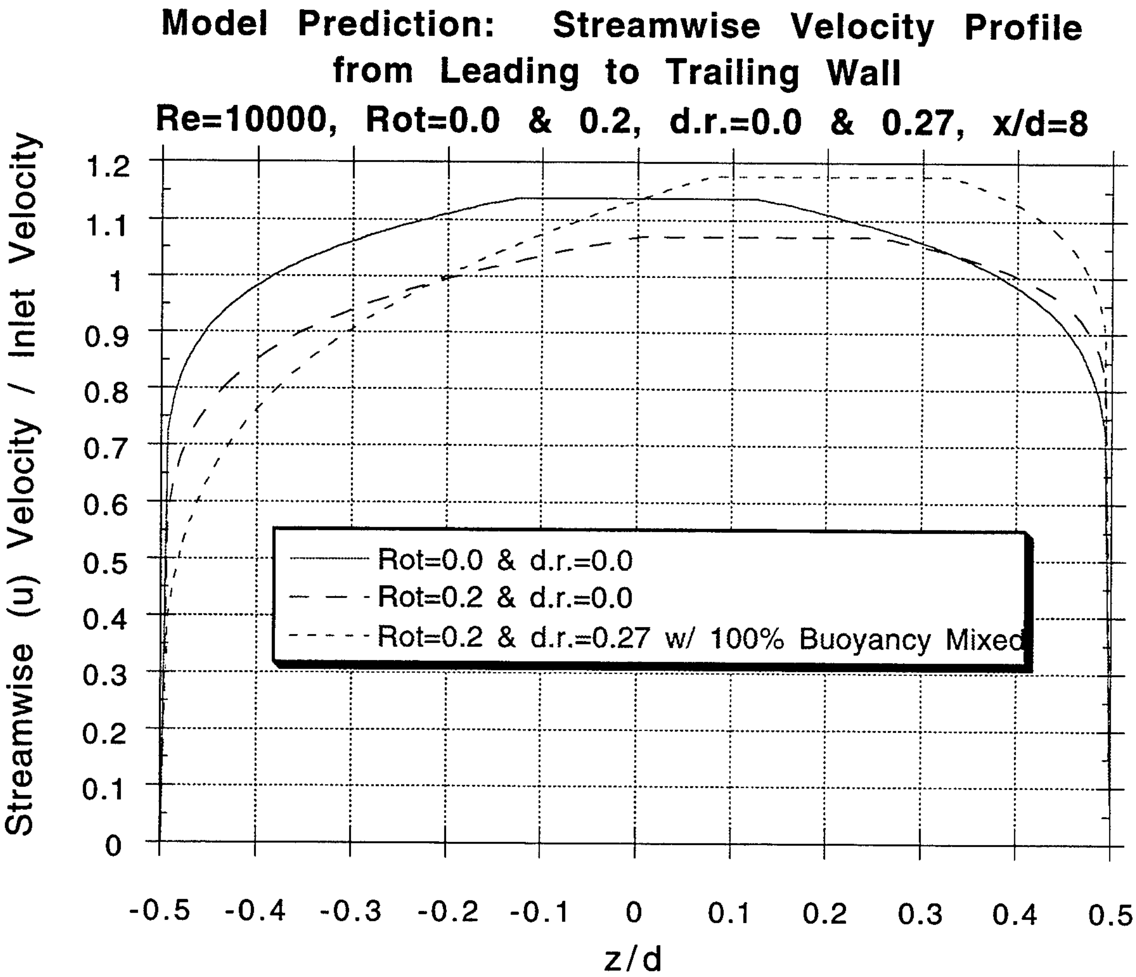


Figure 3.8: Model Prediction of Buoyancy Effect on Leading to Trailing Wall Streamwise Velocity Profile at Midpassage (y/d=0.5)

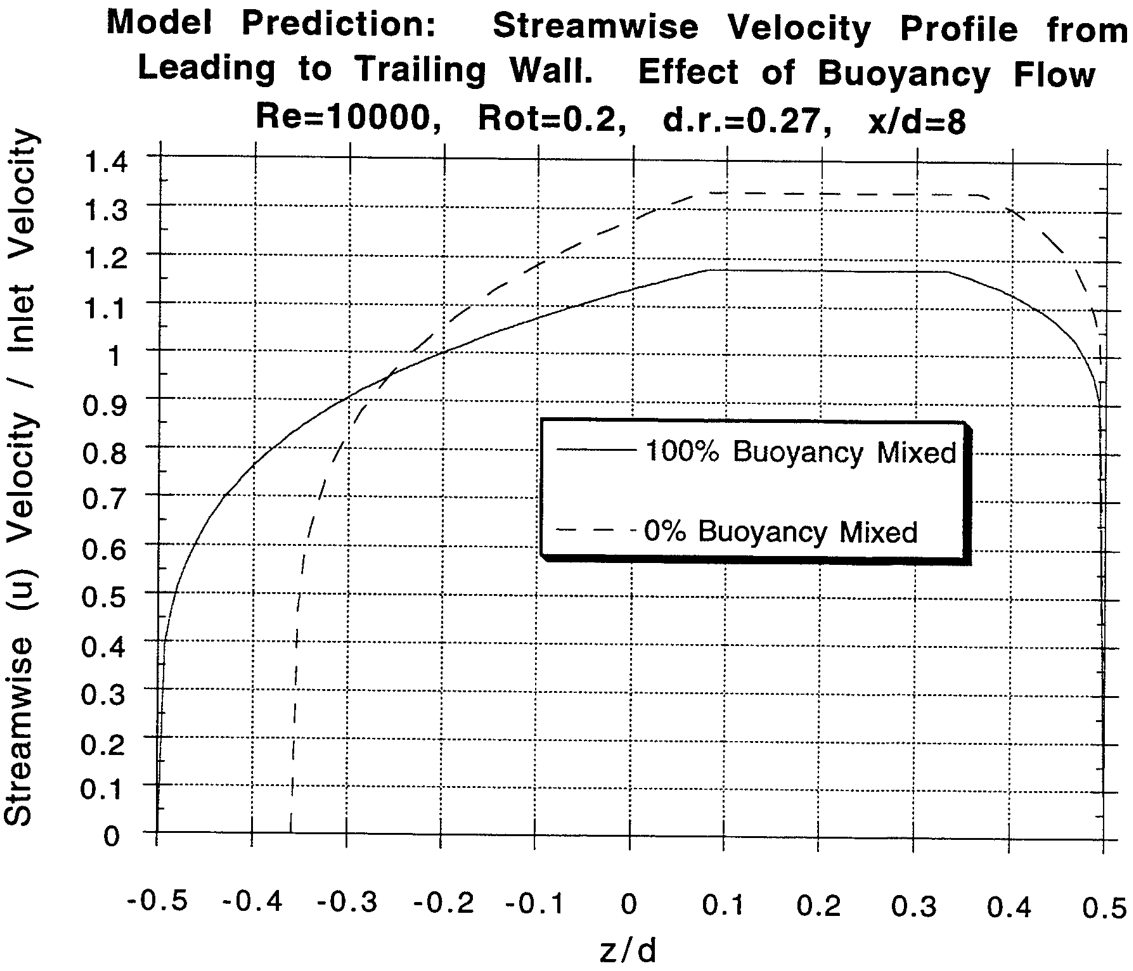


Figure 3.9: Model Prediction of Buoyancy Effect on Leading to Trailing Wall Streamwise Velocity Profile at Midpassage (y/d=0.5). Effect of Different Methods for Treating Buoyancy Flow (100% Mix into Boundary Layer vs. 0% Mix)

Chapter 4: Results & Discussion

Introduction

The experimental data presented in this chapter are in essentially two forms. First there are the PIV fluid velocity measurements and second the passage wall temperature maps (converted to local Nusselt number). The velocity vector field data is presented in several different ways. The orientation of the light sheet allows access to a plane normal to the axis of rotation (x-z plane, see Figure 2.5). Since the strongest velocity gradients are expected in the y and z (non-radial) directions the vector field from each PIV image is first averaged along the z direction (yielding u(z) and w(z) for each image). Due to observed randomness in the vectors, four to six images at each location are combined to produce ensemble-averaged u(z) and w(z) distributions at a given x and y position. The light sheet and camera are then translated in the y and x direction to obtain ensemble averaged u(z) and w(z) distributions at various positions in the flow. u(z) and w(z) at the same x but various y positions can be "stacked-up" to provide a quasi-3D look at the flowfield (assuming the mean flow is steady). Then u(y) and w(y) distributions can be generated at this same x location by interpolating between images in the y direction.

Alternatively, each image can be reduced to a single area averaged vector (averaged in both x and z directions). When lateral (z) variations are minimal (as is the case for stationary flow), these single vectors can be combined to form ensemble-averaged u(y) and w(y) profiles for model comparison. Finally, select vector fields are used to illustrate spatial trends and flow unsteadiness.

Because the model developed in chapter 3 is extremely helpful as a tool to interpret the velocity data, model calculations and experimental data are presented together for stationary, rotating, and heated, rotating cases in that order. The stationary data show the developing boundary layers and accelerating core flow, thus validating the fundamental model assumption made in chapter 3. The rotating data show the Coriolis vortex growth and spatial extent. Data are presented at 2 values of Rot to demonstrate the non-linear dependence of the streamwise Coriolis velocity, u_{cor} from Eqn. 3.6, on Ω. Finally, rotating data with heat show the significant effect of buoyancy on the leading wall. Coupling between the Coriolis vortex and the buoyant boundary layer fluid create a strong adverse effect on the leading side heat transfer. Correlations between measured fluid velocities and heat transfer are made at the conclusion of this chapter.

Stationary Velocity Data

Figure 4.1 shows u(y) data for Re=10000 at two x/d locations. This plot is made using the "single-vector/image" method, so each data point represents the ensemble average of 4-6 images where each image is represented by a single area-averaged vector. The model calculations are also indicated on the figure. Though only four y-positions are available, both the features of boundary layer development and core acceleration can be seen. The 1/9th power velocity profile used in the model matches the data satisfactorily. The corresponding data for w(y) (Figure 4.2) show no significant transverse component when the passage is not rotating. Figure 4.3 shows typical velocity vector fields from the centerline (y/d=0.5) and near wall (y/d = 0.06) data at x/d=7.1. The area-average velocity in both x and z directions are indicated. The near wall vector field shows a lower mean streamwise velocity, but greater spatial variations (turbulence) than the centerline data. Both of these are evidence that the laser sheet at y/d=0.06 has penetrated the side wall boundary layer. This is important because from chapter 3, we expect the Coriolis vortex to be confined to this same region.

Rotating, Non-Heated Velocity Data

Data taken at the same Reynolds number and x/d positions as the above stationary data, but with Rot=0.13 are shown in Figures 4.4 and 4.5. Figure 4.4 shows the measured w(y) data along with model calculations at the same conditions. From the data, it is clear that a secondary vortex flow is present. The w velocity closest to the wall is <0, and a region with w>0 is evident between this and the passage center. The strength and spatial extent of this vortex grow noticeably with x/d. The peak negative w velocity has increased by a factor of 3 with a doubling of distance from the passage inlet (x/d=3.8 vs. 7.1). Also, the crossover point to positive w occurs near y/d=0.13 at the downstream position vs. 0.07 for the upstream position. This is the effect of the Coriolis acceleration component in the z-momentum equation. The model calculations also show this vortex growth. The calculation nearly matches both the peak negative w velocity and the measured crossover point of the experimental data. The profiles selected for the cross-flow are only approximate however, and in the region where w>0 the match is less impressive.

The x/d=3.8 data has a particularly strong w>0 region from 0.1<y/d<0.4 and exhibits a negative w velocity at y/d=0.5 which matches the near-wall level.

While this irregularity was not investigated thoroughly, its origin can be postulated using the same arguments employed in the model development (chapter 3). Since the mechanism for Coriolis secondary flow development is streamwise velocity deficit (Eqn 3.5 repeated here),

$$\Delta w_{cor} \cong 2\Omega\left[\frac{u\rho}{u_\infty \rho_\infty} - 1\right]\Delta x$$

any source of $u<u_\infty$ will result in secondary flow acceleration. In the previous chapter, the boundary layer was treated as the exclusive source of $u<u_\infty$ fluid. However, near the passage inlet local velocity defects may exist due to flow non-uniformities originating upstream of the inlet. This is perhaps the mechanism explaining the w<0 at y/d=0.5 and the strong w>0 region in the x/d=3.8 data. Notably, by x/d=7 this secondary flow has disappeared in favor of the traditional Coriolis double vortex. To avoid such inlet effects and focus exclusively on the passage vortex, all subsequent data was taken at x/d>7.

Figure 4.5 shows u(y) at the same two locations. The most notable feature here is the absence of a boundary layer compared to the non-rotating case. The axial velocity is essentially flat from y/d=0.05 to mid-passage with only a modest rise beyond y/d=0.2. In addition, the core acceleration with x/d that was evident in the stationary case is negligible here. While the model is not as quantitatively accurate here, it does aid in understanding these curious results. The model shows virtually no growth in the side wall boundary layer thickness with x/d and only a modest core acceleration of just less than 10% at x/d=7.1 vs. nearly 13% for Rot=0.0. The mechanism responsible for this result is the streamwise (x-momentum) component of the Coriolis acceleration. From chapter 3 we recall that this acceleration produces additional momentum in the near wall region where w<0 and depresses the velocity in the core region (Eqn 3.6 repeated here).

$$\Delta u_{cor} \cong -2\Omega\frac{w\rho}{u_\infty \rho_\infty}\Delta x$$

The figure 4.5 data show these two effects, though more pronounced than the model calculation. As further proof that u_{cor} is indeed the responsible mechanism, Figures 4.6, 4.7, and 4.8 show data for a higher rotation number (Rot=0.2 and Re=8100 at x/d=7.9). Here the model calculations are plotted with and without the streamwise Coriolis effect (both incorporate the z-momentum component). Figure 4.6 shows a dramatic rise in u as the wall is approached rather than the flat profile witnessed for Rot=0.13, evidence that the streamwise Coriolis acceleration is the culprit. From equation 3.6, it is clear that u_{cor} is proportional to Ω^2 rather than Ω since w_{cor} is itself proportional to Ω. A 60% rise in rotational frequency (from Rot=0.13 to 0.2) actually produces a rise in u_{cor} by a factor of 2.4. The core depression is also dramatic, where the model-calculated core acceleration drops from 15% to 7% with the addition of u_{cor}.

Figure 4.7 shows the w(y) data and the model calculation (again with and without u_{cor} incorporated). The effect of the Coriolis streamwise component is considerable. Since u_{cor} energizes and thins the side wall boundary layer, the Coriolis vortex flow is reduced in its spatial extent (which is a function of the side wall boundary layer thickness). This squeezes a reduced secondary mass flow into a narrower region near the wall and increases the peak velocity in the vortex. The addition of u_{cor} to the model drops the w=0 position from y/d=0.14 to 0.07 and lowers the minimum w from -0.37 to -0.47. Both of these trends bring the model calculation more in line with the measured results. Finally, Figure 4.8 shows the u(z) calculation vs. the experimental data. The boundary layer disparity between the trailing and leading walls is apparent. The vortex motion has thinned the trailing wall boundary layer and thickened the leading wall boundary layer. This is the convection of boundary layer fluid modeled as $\Delta\delta_d$ in chapter 3. By incorporating the boundary layer adjustment in this manner, the calculation result matches the experimental data almost exactly.

Other researchers have noted these same flow features to varying degrees. Berg et al [4] measured the velocity field in a long l/d rotating pipe (Re=40000 and Rot=0.04) with laser 2-focus and found a rise of 10% in the streamwise velocity from passage centerline to sidewall and a trailing-to-leading wall edge velocity ratio of 1.15. Tse and McGrath's [48] LDV data show secondary velocity magnitudes up to 65% of the streamwise velocity at x/d=7 for Re=25000 and Rot=0.24. The streamwise velocity rise toward the side walls is also evident in their data as is the predicted second vortex pair. The PIV data presented in this thesis show no evidence of a second vortex pair. Since the second vortex pair's location is predicted to be near the leading wall, where optical access is difficult, perhaps the vortex exists but is not captured by this application of PIV.

Typical velocity vector fields from the centerline ($y/d \approx 0.5$) and near wall ($y/d \approx 0.05$) rotating data are shown in Figure 4.9 for this Re and Rot. The near wall vectors show the strong motion in the direction of rotation vs. the centerline vectors which are leaning in the opposite direction. The y/d=0.93 vectors also exhibit the steep dw/dy gradient captured by the finite laser sheet thickness. Vectors with w/u_{in} = -0.5 lie side by side with w/u_{in} = 0.0 vectors in this 2D representation of a 3D flowfield. With data always plotted in ensemble averages, it is natural to form the opinion that the observed secondary flow is stationary with respect to the passage. To dispel this erroneous notion, Figure 4.10 shows three sequential images at y/d=0.91 (precisely at the vortex eye). Each image is separated by one rotor revolution (0.1 seconds), or a net fluid motion of 30d. The laser sheet is now located at the point of steepest gradient in the crossflow shear layer, thus the wide range of seemingly random vectors pointed toward and away from the rotation direction. Again due to the finite (0.4mm) sheet width, particles moving in both directions are captured in the same 2-D image plane. The area-averaged u is roughly constant for all 3 images, but the averaged w varies from -0.61 to -0.13. This suggests that either the vortex is moving in the y direction with time or that the vortex strength is not steady. More research is necessary to fully document this finding and determine its effect (if any) on the steady state wall Nu.

Two additional figures exhibit details of the rotating flowfield not captured by the integral model. Figures 4.11 and 4.12 show two different viewpoints of the combined effects of Coriolis on the streamwise velocity. The first figure shows u(z) at 3 different regions of y. To create this plot the u(z) distributions for all images in a specified region of y are averaged together to form one distinct profile. The profile for $0.29<y/d<0.56$ exhibits the disproportionately large thickness of the leading wall boundary layer relative to the trailing wall. As y moves toward the wall, the profile becomes flatter and finally rises showing the strong acceleration due to the Coriolis streamwise component (u_{cor}). The effect of u_{cor} is clearly strongest for $z<0$ since the Coriolis vortex is accelerating flow from the trailing wall to the leading wall. The vortex strength is largest for $z<0$ and thus the corresponding streamwise effect of Coriolis is more intense here as well. Figure 4.12 shows the same trend with now u(y) for various regions of z. Here again as z decreases the core region velocity drops (due to the large leading wall boundary layer) and the wall region velocity rises (due to the streamwise Coriolis acceleration).

Rotating, Heated Velocity Data

The final experimental step was the addition of surface heat flux, creating a non-isothermal flowfield and its associated buoyancy acceleration in the rotating reference frame. While the non-heated data above provide insight into the significance of Coriolis effects on boundary layer development and secondary flow, a cooling passage is by definition non-isothermal. Thus heated, rotating velocity measurements are needed to fully characterize the flowfield. These data, correlated with the associated heat transfer data, represent the most important contribution of this work.

Figure 4.13 shows the most striking result of heat addition. Plotted are u(z) distributions from the 3 cases: Rot=0.0 & d.r.=0.0, Rot=0.2 & d.r.=0.0, and Rot=0.2 & d.r.=0.27. With heat addition, the trailing to leading wall disparity in streamwise velocity exceeds a ratio of 2. This dwarfs the same measure from the non-heated data, shown in Figure 4.8 (and again in this figure) to be only 1.1 . Since Rot is identical in the two data sets, the distorted streamwise velocity profile for the heated case is clearly the result of hot, buoyant fluid collecting on the leading wall. To substantiate this assertion, again we can resort to the integral model. As outlined in chapter 3, the model accounts for buoyancy-driven flows with two distinct strategies. The first strategy allows for 100% mixing of the associated mass defect into the turbulent boundary layer. The second strategy assumes a separation bubble is created on the leading wall whose spatial extent is dictated by the net mass defect in the buoyant flow. Figure 4.14 compares calculations using these two methods with the PIV velocity data. While both methods show a strongly distorted velocity profile, only the separation method comes close to matching the measured level of core acceleration.

Figure 4.15 is a surface map of the PIV streamwise velocity data. Clearly a large portion of the test section passage has fluid velocities greater than the normalizing inlet velocity. A rough continuity check on the measured surface map indicates that there must be a very low streamwise velocity region near the leading wall. This and the model comparison in Figure 4.14 combine to strongly suggest stagnant or reverse flow near the leading wall. Unfortunately, as mentioned earlier, it is difficult to optically image the near wall region on the leading face of the test section, but isolated vectors from a few selected PIV images do indicate the presence of stagnant or reversed flow. Figure 4.16 shows actual vector fields where the u(z) gradient and sites of stagnant flow along the leading wall are evident. Again, to show that the position of this buoyant flow effect is not completely steady, Figure 4.17 shows three sequential vector fields at y/d=0.67. The region of low velocity toward the leading side changes noticeably between images, suggesting unsteady motion in the z direction.

Since there is no heated rotating velocity data to compare this new data with, the results of several computational studies which give evidence of similar flow features are presented for comparison. Dutta et al [13] predicted a heated, rotating flowfield based on the UTRC configuration at Re=25000, Rot=0.24, and d.r.=0.13 at x/d=9. The code results show a streamwise velocity of $0.7u_{in}$ at z/d=-.35 and $1.2u_{in}$ at z/d=+0.35. Also, vectors near the side walls are shown with secondary velocity magnitudes near $0.15u_{in}$ and an estimated vortex eye position of y/d=0.1. For the same conditions, but a lower Rot=0.12, Tekriwal [46,47] predicted nearly a factor of 2 in streamwise velocity from trailing side to leading side and shows cross-stream velocities near 0.2 u_{in} with a vortex eye position of y/d=0.15. At a higher Rot=0.48, Tekriwal predicted reverse flow over a region extending to Δz/d=0.15 from the leading wall. Prakash and Zerkle [45] also predicted reverse flow for this same high Rot case. Finally, Bonhoff et al [6] predicted reverse flow and a peak streamwise velocity of $1.8u_{in}$ near the trailing wall for Rot=0.24, Re=25000, and d.r.=0.15. Though the three codes all predicted similar results (reverse flow and core velocity skewed to trailing wall), the published data is for three very different passage locations: x/d=8 for Tekriwal, x/d=2 for Prakash and Zerkle, and x/d=14 for Bonhoff et al. The data presented here are at an x/d=8, a lower Re=10000, a moderate Rot=0.2, and a factor of two higher d.r.=0.27 from the above predictions. While a CFD prediction reproducing these exact flow conditions is not yet available, this comparison shows that the measured behavior is in the range of that previously predicted.

That the Coriolis vortex is still as vigorous in this heated environment is evident from Figure 4.18. In fact, the position and strength of the w<0 flow near the wall appear to be relatively unchanged from the d.r.=0.0 case. Data is shown over the entire test section width for this case, so the flow symmetry is evident. It is this vortex which creates the disproportionate effect of buoyancy on the leading wall. The hot wall fluid from the trailing and side walls all collects on the leading wall due to the convective action of the swirling vortex. As with the non-heated case (Fig 4.7), the model flow calculation captures the magnitude and location of the near wall part of the vortex but falls short in the region where w>0. This indicates more rapid diffusion and mixing in the core region of the passage than incorporated in the model.

The other Coriolis effect on the streamwise velocity is also present in the heated PIV data, though now it competes directly with the buoyancy force on the side walls. Figure 4.19 shows u(y) profiles at various z stations. Near the back side wall (y/d=0), the streamwise Coriolis component creates a strong acceleration at z/d = 0.3 which then decreases with z. This acceleration is not as evident on the front side wall (y/d=1) until z/d = -0.2. This is due to the heating imbalance mentioned in chapter 2. Due to a manufacturing error, the front side wall has a higher heat flux than the back side wall. Because of this, buoyancy has a stronger effect on the front side wall than the back side wall, and streamwise velocities along this wall (y/d=1) are lower. Upon closer inspection of the near wall secondary velocities (w) in Fig 4.18, the peak negative velocity on the front side wall (y/d=1) is 20% higher than on the back side wall (y/d=0). Also, the region of positive w velocity extends further from the front wall than from the back side wall.

Since the boundary layer to core fluid density ratio also figures into the w_{cor} calculation of equation 3.5, the stronger vortex on the front side wall is again evidence of the heat load disparity between the two side walls. This combination of stronger vortex and greater buoyancy sharply reduce the heat transfer at the front side edge of the leading wall as will be seen shortly. Figure 4.20 also shows the asymmetry in the velocity data. Here u(z) is plotted for various y regions. The region of highest velocity is $0.1<z/d<0.3$ and $y/d=0.1$ (near the back side wall). The region of lowest velocity is $-.3<z/d<-0.1$ and $y/d=0.86$ (near the front side wall). More hot gas has collected in the front side/leading wall corner while conversely the back side/trailing wall corner is less effected by buoyancy.

Nusselt Prediction Interlude

Before presenting the Nusselt map data for the above case, it is instructive to attempt a heat transfer prediction based on the velocity data alone. First, as regards the leading and trailing walls, ignoring for the moment that the data suggests separation on the leading face due to buoyancy, we could estimate the Nusselt number for each side based solely on their associated boundary layer edge velocity. From Figs 4.13 and 4.20 this ratio appears to be approximately 2.2 (trailing wall to leading wall respectively). Following the analytical Dittus-Boelter [11] correlation between Re and Nu, we would expect

$$\frac{Nu_{TS}}{Nu_{LS}} \propto \left\{\frac{Re_{TS}}{Re_{LS}}\right\}^{0.8} = \left\{\frac{u_{\infty TS}}{u_{\infty LS}}\right\}^{0.8} \cong 2.2^{0.8} = 1.88$$

As will be shown subsequently, this is close to the experimentally measured Nu ratio at this x/d.

Turning now to the front side and back side walls, from figures 4.19 and 4.20 the streamwise Coriolis effect is more pronounced on the back side wall as buoyancy is a more significant factor on the front side wall. So, one might predict a slightly higher Nu along the back side wall. However, the vortex strength is greater on the front side wall than on the back side wall (Fig 4.18) due to the higher heat load and higher fluid temperatures here. Assuming that the stronger vortex could more than make up for the adverse effect of buoyancy on this wall, we might predict a higher Nu on the front side wall than on the back side wall. This latter result turns out to be true, suggesting that the vortex does play the dominant role in convection along the side walls.

Rotating, Heated Nusselt Measurements

The actual experimentally measured quantities here are wall temperature (via the infrared scanner) and net current through the ITO film on the four test section exterior walls. From this, the temperature and heat flux distributions on the inside walls are calculated as described in chapter 2. Finally, Nu is computed as,

$$Nu = \frac{q_w/(T_w - T_{film})}{k_{film}/d}$$

The plots of Nu shown here are normalized by the Dittus-Boelter [11] correlation for Nu in a stationary tube.

$$Nu_\infty = 0.023\,Pr^{0.4}\,Re^{0.8}$$

This is done to remove the Reynolds number dependency from the data. As mentioned in chapter 1, the experimental data in the literature provides no consensus as to whether this is successfully accomplished or not [16,17,51]. Data is presented for $0.02<Rot<0.29$ all at $Re\approx8200$ and d.r.≈0.27.

Figure 4.21 shows the Nusselt ratio contours on the inside walls of the test section for the same conditions used in the heated PIV study just presented. The four walls are displayed in succession from left to right: front side wall ($y/d=1$), leading wall ($z/d=-0.5$), back side wall ($y/d=0$), and trailing wall ($z/d=+0.5$) (see Figure 2.5). A low Nu spot is evident on the leading wall at $x/d=8$ and a corresponding high Nu region is located on the trailing wall at the same x/d. The Nu on the other 2 sides appears to vary monotonically between the leading and trailing wall peak values. Upon closer inspection, the low Nu region on the leading wall is skewed to the front side wall rather than the back side wall and the lateral Nu gradient on the front side wall appears steeper than that on the back side wall. This is evidence of the disparity in surface heat flux between the two walls. The increased heat addition on the front wall produces more buoyant hot fluid which collects on the leading wall creating the skewed Nu map. Also, the vortex is stronger along this wall causing a steeper Nu gradient than on the back wall.

The magnitude of these two effects is more clearly evaluated with Figure 4.22 which shows the side mean Nu as a function of x. As expected $Nu_{TS} > Nu_{LS}$ by nearly a factor of 2 at x/d=8. This nearly matches the edge velocity prediction made earlier. While Nu_{TS} climbs nearly linearly from inlet to exit, Nu_{LS} is flat until x/d=5 and then drops to a low at x/d=8. As the velocity data indicated, the buoyancy effect is considerable at x/d=8, creating a huge disparity between the flowfield on the leading wall vs. the trailing wall. If a separation bubble on the leading wall is the flow feature causing this low Nu at x/d=8, it is logical to pick a separation point based on the Nu data alone at approximately x/d=5 (since this is where Nu_{LS} begins to drop). Extrapolating further, there may be a flow reattachment at x/d=9, as Nu_{LS} begins to rise here. x/d=9 is also near the passage exit (l/d=11.5), so this rise may be due to a recirculating flow from the exit plenum.

The comparison of the front and back wall Nu ratios is more complex. On the average, Nu_{FS} is 15% greater than Nu_{BS}. The greater heat load added on the front side appears to have the primary effect of strengthening the vortex here, thus creating an increased convective heat transfer and higher Nu. There are also noticeable oscillations in the Nu data. Nu_{FS} initially rises with Nu_{TS} but drops down coincident with the drop off in Nu_{LS} at x/d=5. Then, there is a reprieve from x/d=6 to 8 where Nu_{FS} rises and Nu_{BS} drops. Finally, Nu_{FS} drops to the exit and Nu_{BS} overtakes Nu_{FS}.

To get some idea as to what may cause these apparent oscillations, Fig 4.23 shows plots of Nu(y) at various x locations on the leading wall. Clearly the Nu_{LS} profile flattens around x/d=8, while before and after this point Nu_{LS} is lower near the front side edge. One possible explanation is the collection of a strong buoyancy-generated separation bubble near the front-side/leading wall corner due to the heating imbalance. As the bubble grows with x/d it slowly migrates laterally until it fills the whole passage width (creating the flat Nu_{LS} profile near x/d=8). At this point, no further migration is possible and buoyant fluid again collects disproportionately on the front side/leading wall edge. The result is a reduced Nu here after x/d=8. Han and Zhang [17] found that when a wall-to-wall heating imbalance (uneven wall temperatures) was imposed on a rotating coolant passage, Nu was higher on all of the passage walls compared with the constant wall temperature case (at the same density ratio). They postulated that the buoyancy imbalance destabilizes the core flow, creating lateral motion like that seen here. Another explanation could be the presence of a characteristic "Helmholtz-like" fluid instability due to the buoyant rotating flow, the bulk flow moving side-to-side in the passage alternating between the front and back side walls. Either of these would account for the Nu oscillations in Fig 4.22, but more data is needed to confirm their validity.

Figures 4.24 and 4.25 show the effect of rotation number on the trailing and leading wall Nu ratios. On the trailing wall, Nu increases everywhere with Rot. For the largest Rot=0.28 there are oscillations evident on the trailing wall, which at lower Rot are isolated to the side walls. Though PIV data was taken only for Rot=0.2, trends in the Nu data with Rot can be interpreted by extrapolating from the available velocity data. Perhaps at the highest rotation, a separation bubble grows from x/d=2 (where Nu_{LS} drops in Fig. 4.25) to approximately x/d=5 when it collapses causing flow to move back toward the leading wall. This would explain the flat Nu_{TS} from 5<x/d<7 and the Nu_{LS} increase beyond x/d=5. Beyond x/d=7, it may be that the separation regains significance and Nu_{TS} increases as the remaining passage core fluid is once again pushed away from the leading wall and in the direction of the trailing wall.

From Fig 4.25, the Nu_{LS} data do not vary monotonically with Rot at every x/d. This highlights the competing roles of Coriolis (with components proportional to Ω and Ω^2) and buoyancy (which varies with Ω^2 only). Looking at both figures it is apparent that $Nu_{TS} > Nu_{LS}$ even at the lowest Rot=0.02. Though at Rot=0.02 both Nu_{LS} and Nu_{TS} fall with x/d (to an average of 1), Nu_{TS} is 30% higher overall. Figure 4.26 shows the ratio Nu_{TS} / Nu_{LS} for the same four conditions and Figure 4.27 shows the average value $Nu_{avg} = (Nu_{LS}+Nu_{TS})/2$. Here it is again clear that while Rot=0.02 produces a disparity of $Nu_{TS} > Nu_{LS}$, the passage mean Nu converges to the stationary correlation value. For the other three cases, the passage mean Nu and the trailing to leading wall Nu disparity grow with Rot. This underscores again the complicated nature of this rotating, heated flowfield where buoyancy and Coriolis forces interact with comparable magnitudes.

Data taken by Barry [2] on the same facility show a nearly identical trend for Nu_{TS} / Nu_{LS} , though the Barry data is at Re=25000. The ratio rises to approximately 2 by x/d=8 and drops off toward the exit. The absolute Nu data are 50% higher for the Barry data compared to the current data and the shape of the Nu(x) curves is also different with both Nu_{TS} and Nu_{LS} dropping from inlet to x/d=6 and rising modestly thereafter. Guidez [16] also measured a Nu_{TS} / Nu_{LS} ratio of 2 at x/d=7.4 and Re=24000, Rot=0.2, and d.r.=0.36. Guidez' data, however, did show a drop in Nu_{TS} / Nu_{LS} of 10% for a 50% rise in Re to 33000. Wagner et al [51] show a Nu_{TS} / Nu_{LS} ratio of nearly 3.5 for Re=25000, d.r.=0.22, and Rot=0.2. Both the Guidez and Wagner et al data show Nu_{TS} rising to a maximum at x/d=8 and Nu_{LS} dropping from inlet to x/d=6 (followed by a rise thereafter). Wagner et al also show that Nu_{LS} decreases and then increases with increasing Rot, like that shown in Figure 4.25.

Closure

Though comparisons of the measured Nusselt number in this experiment to that of other experiments is useful, it should not obscure the more important comparisons made earlier. Because both velocity and heat transfer data are available here, one can now do more than postulate the existence of flow structures and their effects on heat transfer. The velocity data show stagnant leading wall fluid and the Nu data are also depressed on the leading wall. The velocity data show a thin trailing side boundary layer and the Nu data are correspondingly high there. The velocity data show a thin side wall boundary layer with high crossflow velocities and the Nu data show no effect of buoyancy on the side walls. The velocity data even show a disparity in vortex strength from the front to back side walls, a result of a manufacturing heat flux imbalance. The Nu data also bear out this imbalance. This is the essential contribution of this thesis, measured flow structures are correlated directly with measured heat transfer to make a prediction like $Nu_{TS} / Nu_{LS} \approx 2$ based on the measured edge velocity possible.

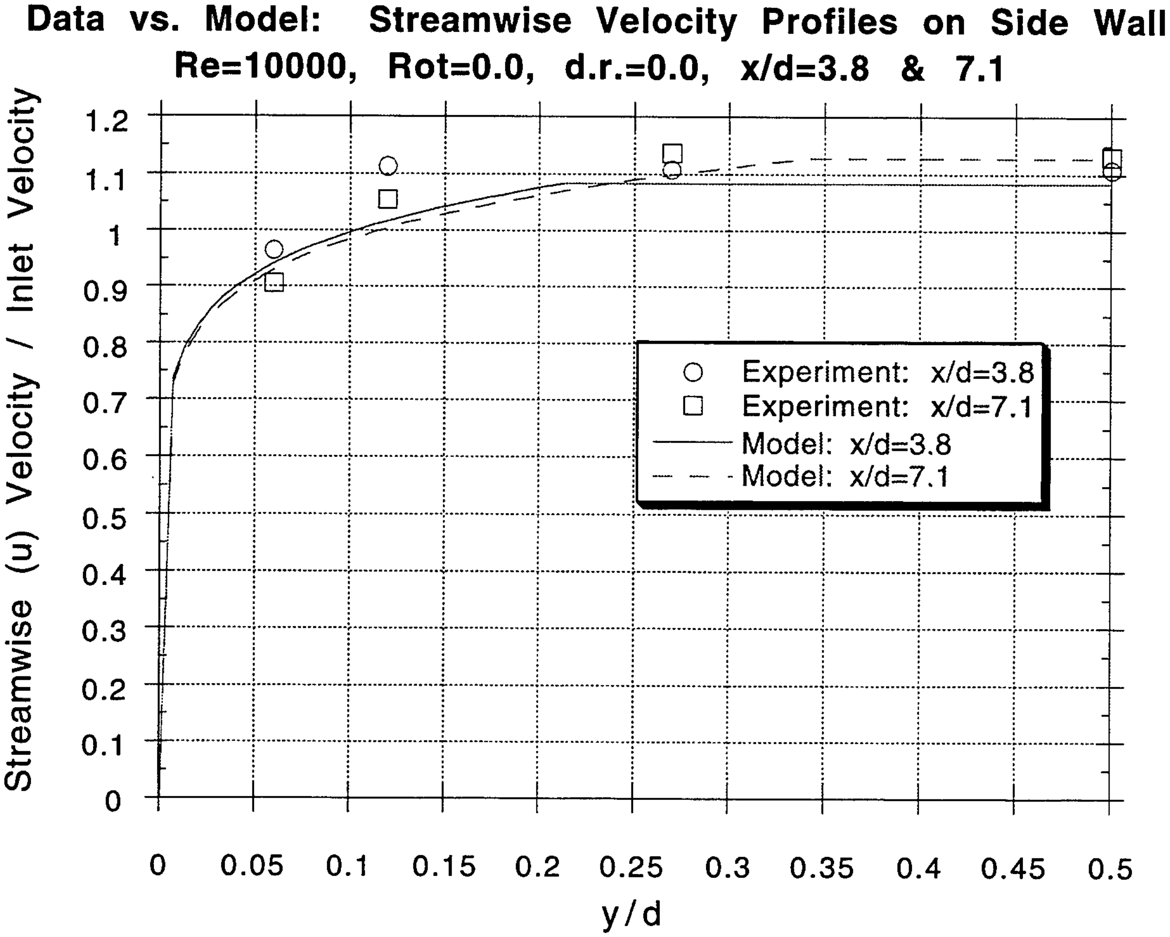


Figure 4.1: Data vs. Model Prediction of Streamwise Velocity Profiles for Rot=0.0

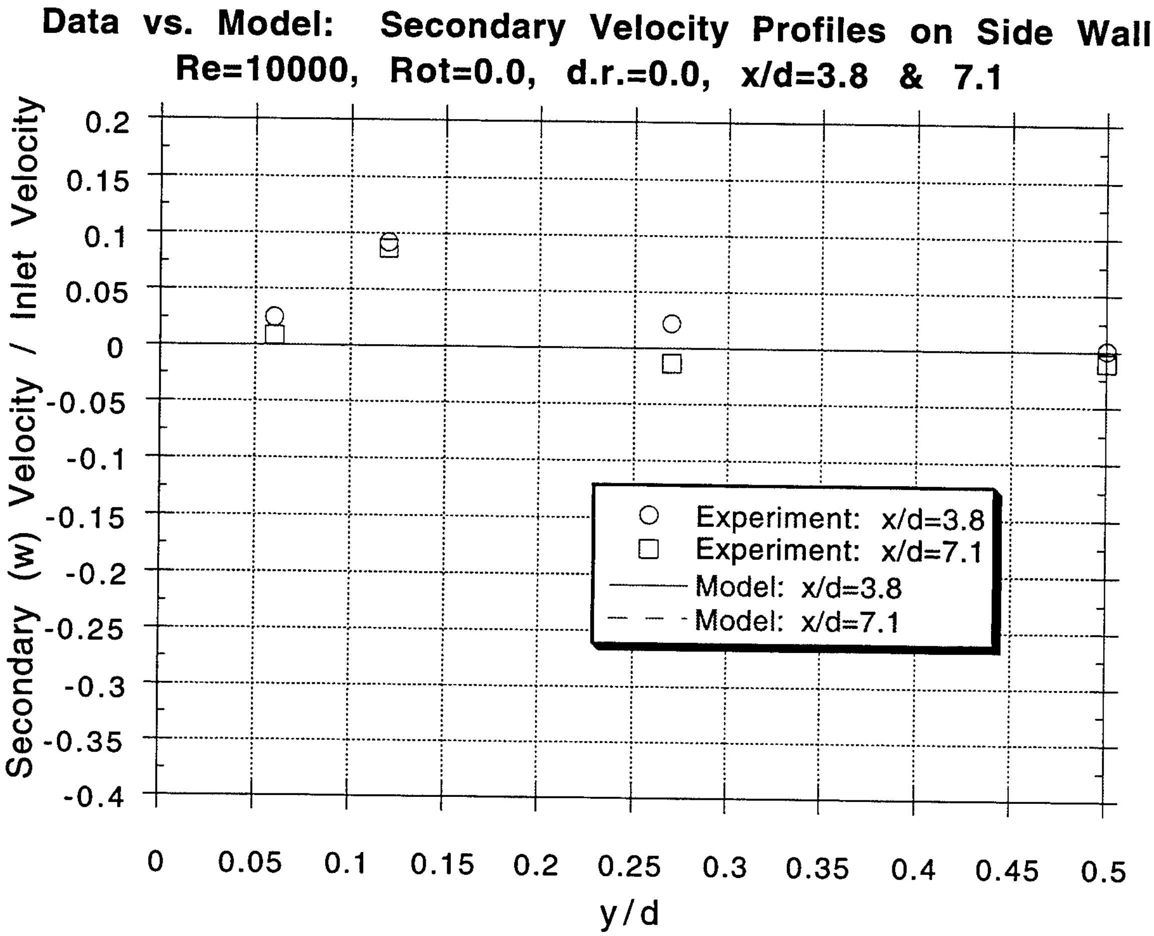
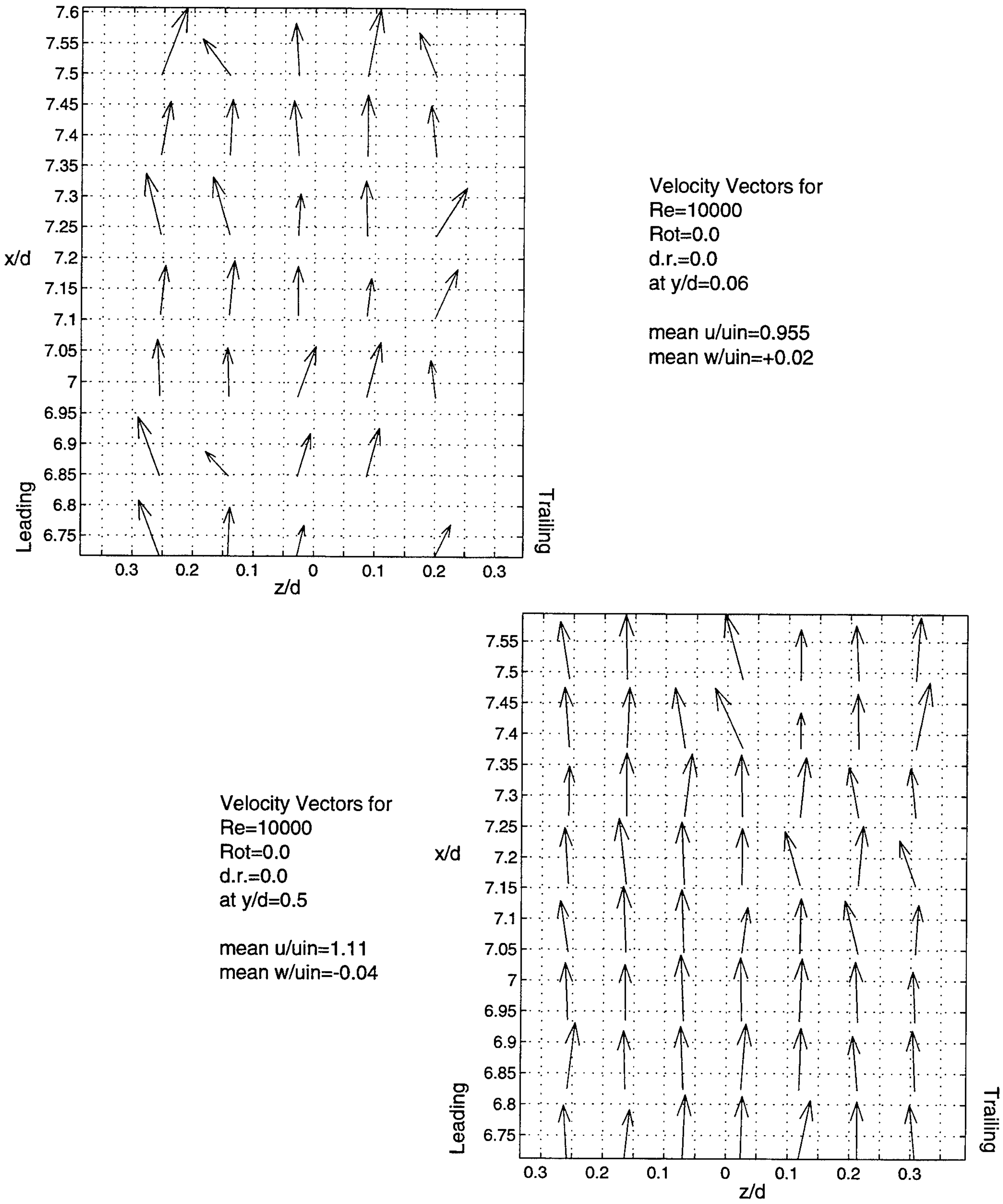


Figure 4.2: Data vs. Model Prediction of Secondary Velocity Profiles for Rot=0.0



Velocity Vectors for
Re=10000
Rot=0.0
d.r.=0.0
at y/d=0.06

mean u/uin=0.955
mean w/uin=+0.02

Velocity Vectors for
Re=10000
Rot=0.0
d.r.=0.0
at y/d=0.5

mean u/uin=1.11
mean w/uin=-0.04

Figure 4.3: Sample Vector x-z Plots for Re=10000, Rot=0.0, and d.r.=0.0 at x/d=7.2

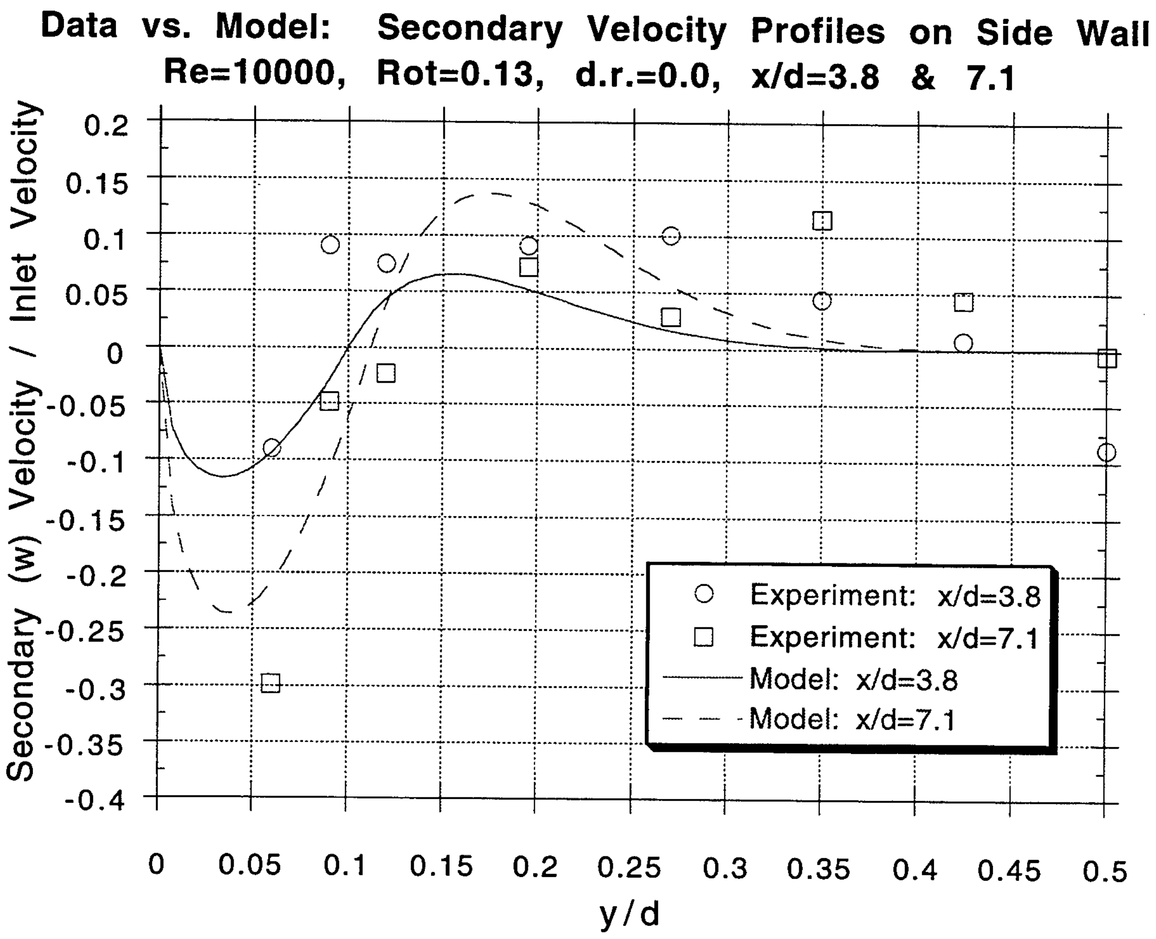


Figure 4.4: Data vs. Model Prediction of Secondary Velocity Profiles for Rot=0.13

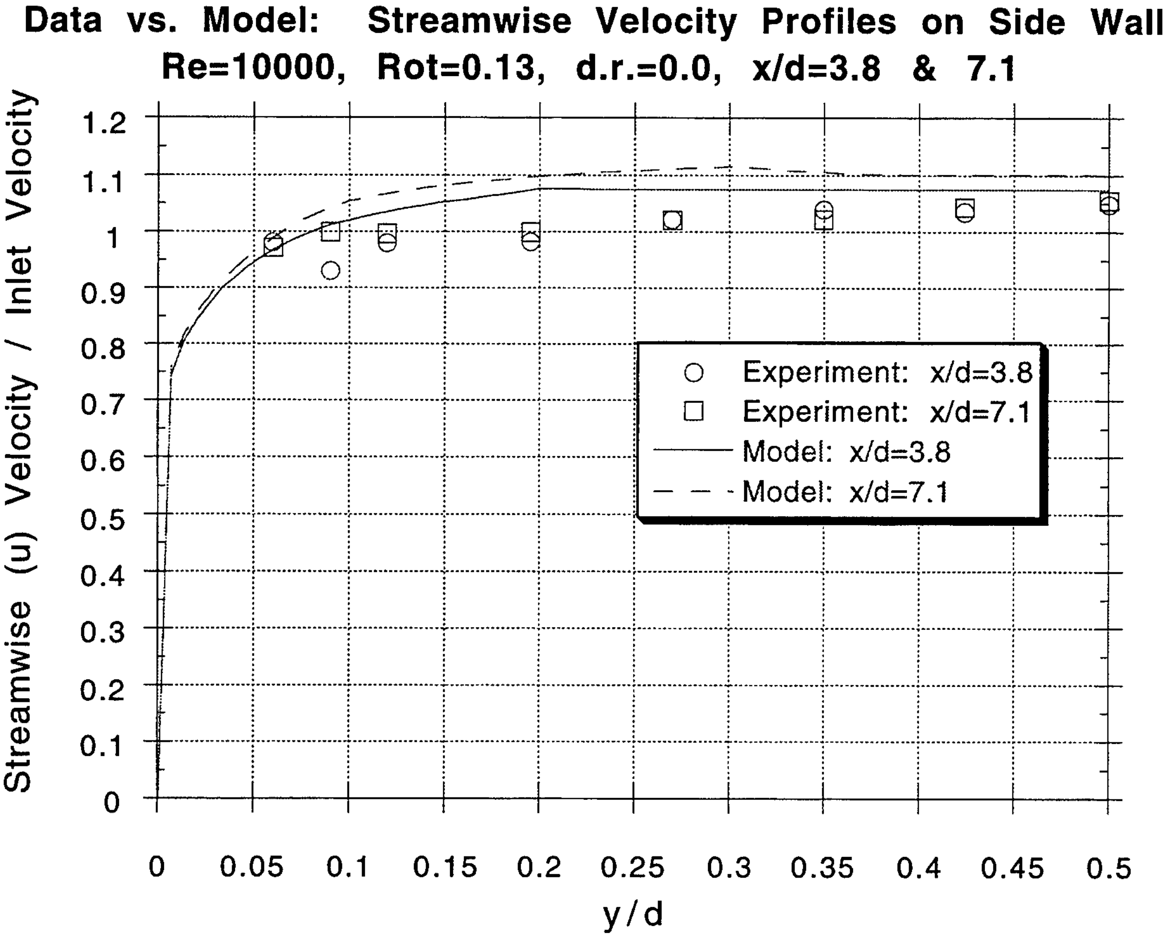


Figure 4.5: Data vs. Model Prediction of Streamwise Velocity Profiles for Rot=0.13

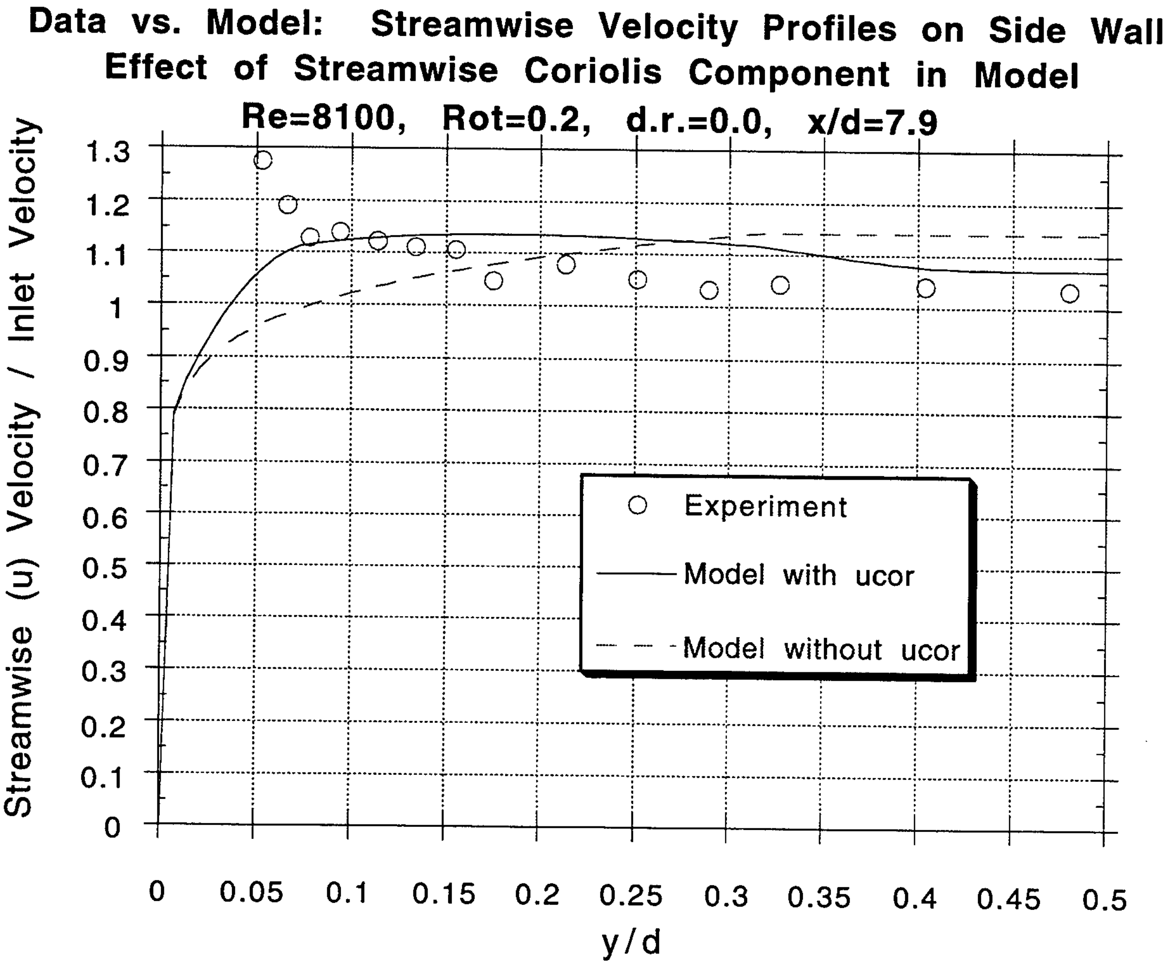


Figure 4.6: Data vs. Model Prediction of Streamwise (y) Velocity Profiles for Rot=0.2
Effect of Streamwise Component of Coriolis Acceleration in Model

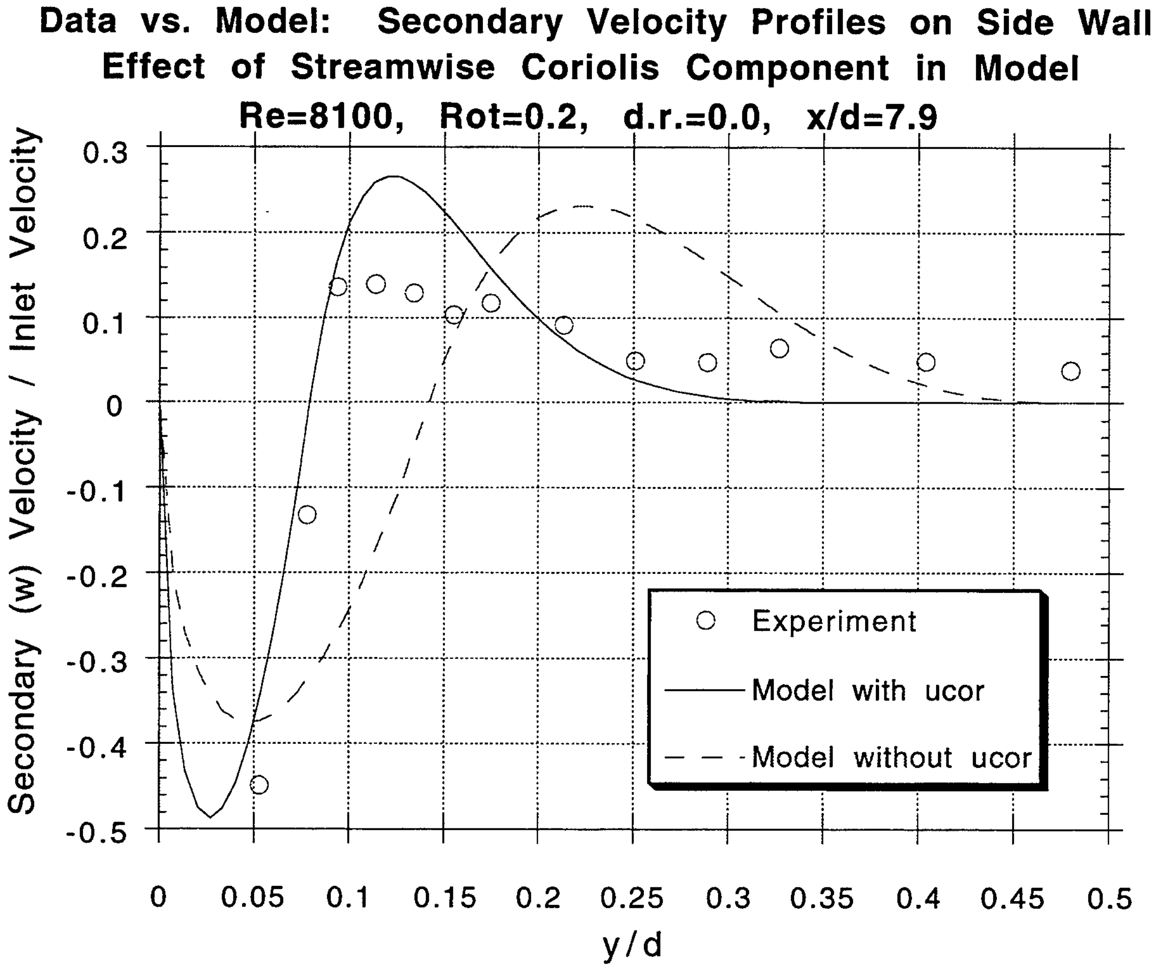


Figure 4.7: Data vs. Model Prediction of Secondary Velocity Profiles for Rot=0.2
Effect of Streamwise Component of Coriolis Acceleration in Model

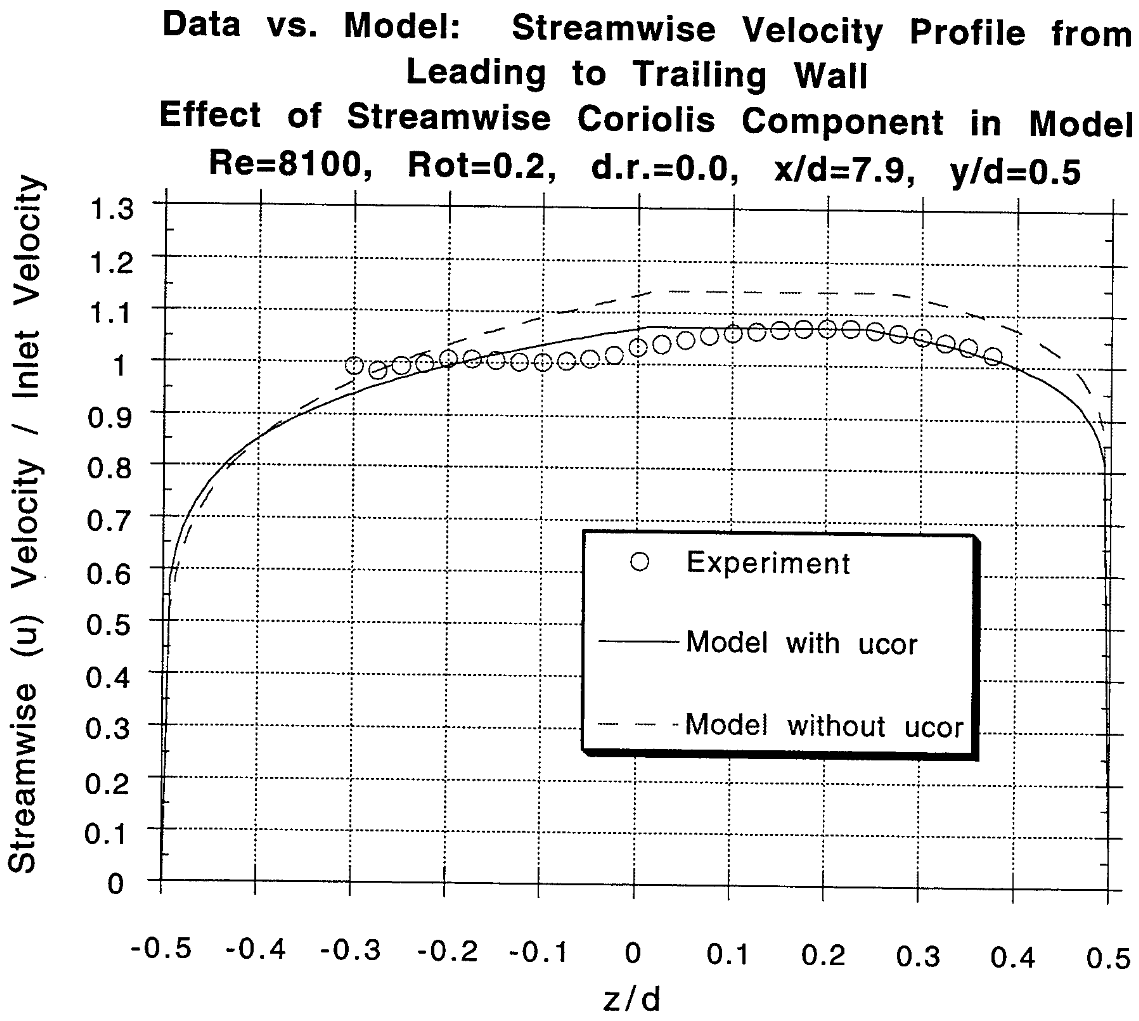
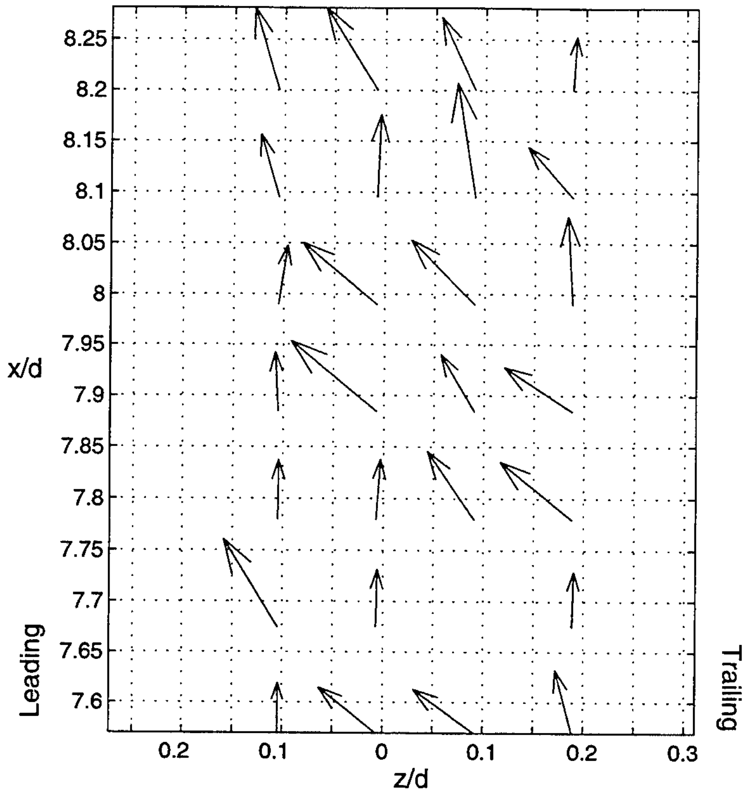


Figure 4.8: Data vs. Model Prediction of Streamwise (z) Velocity Profiles for Rot=0.2
Effect of Streamwise Component of Coriolis Acceleration in Model



Velocity Vectors for
Re=8100
Rot=0.2
d.r.=0.0
at y/d=0.93

mean u/uin=1.21

mean w/uin=-0.57

Velocity Vectors for
Re=8100
Rot=0.2
d.r.=0.0
at y/d=0.58

mean u/uin=1.04

mean w/uin=+0.06

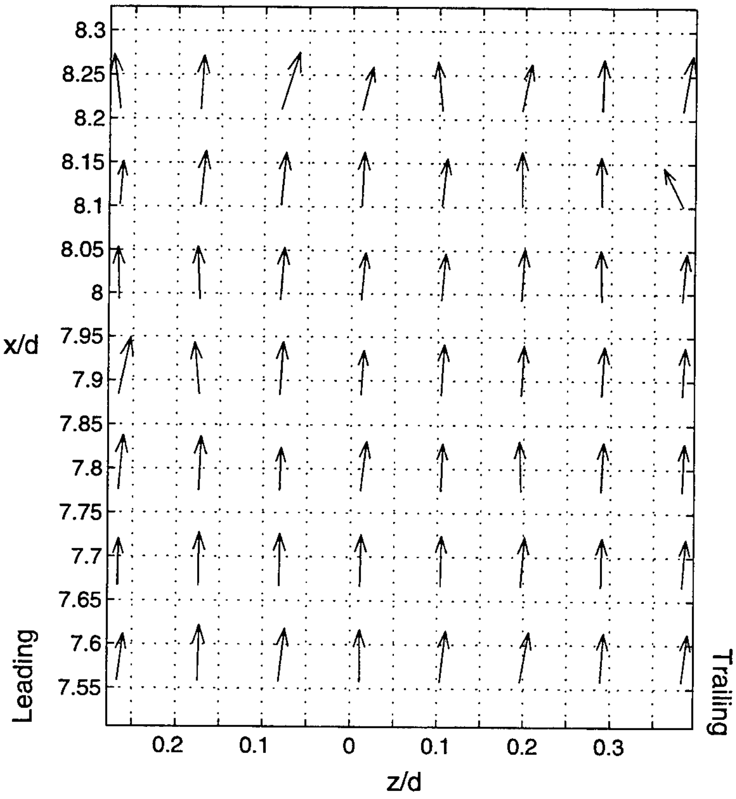
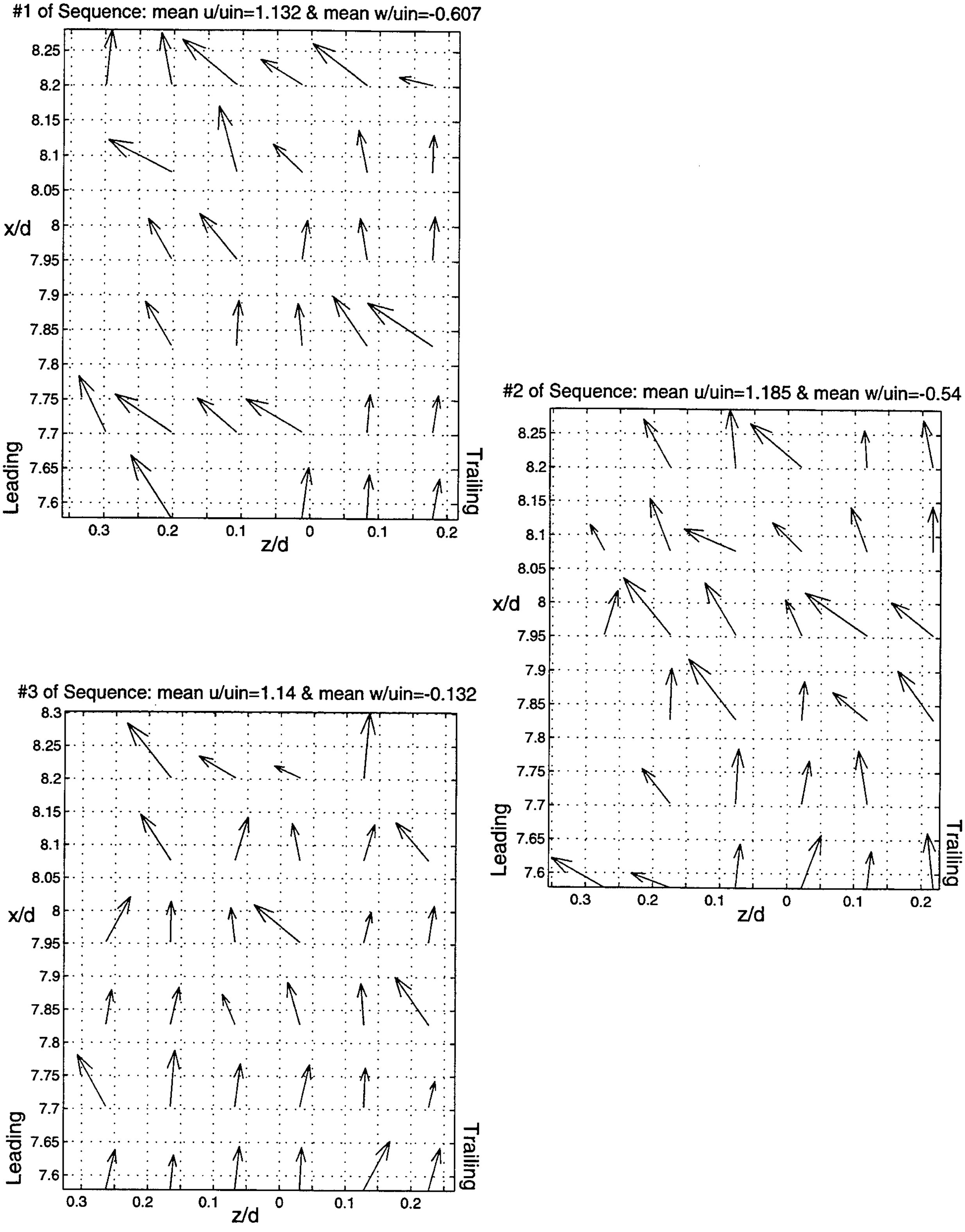


Figure 4.9: Sample Vector x-z Plots for Re=8100, Rot=0.2, and d.r.=0.0 at x/d=7.9


#1 of Sequence: mean u/uin=1.132 & mean w/uin=-0.607

#2 of Sequence: mean u/uin=1.185 & mean w/uin=-0.54

#3 of Sequence: mean u/uin=1.14 & mean w/uin=-0.132

Figure 4.10: 3 Sequential Vector x-z Plots for Re=8100, Rot=0.2, & d.r.=0.0 at y/d=0.91.

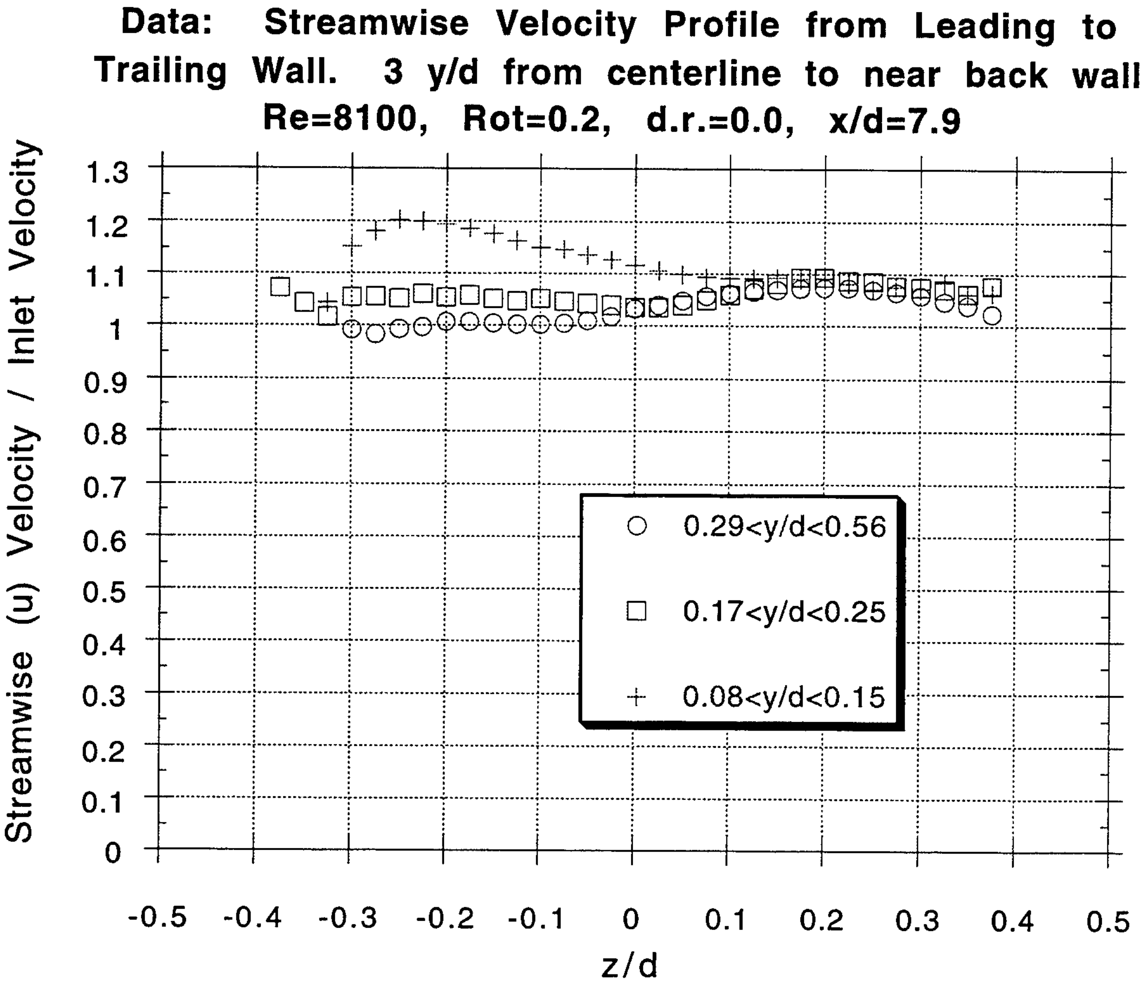


Figure 4.11: Streamwise (z) Velocity Profile Data for Rot=0.2 at 3 y/d Locations

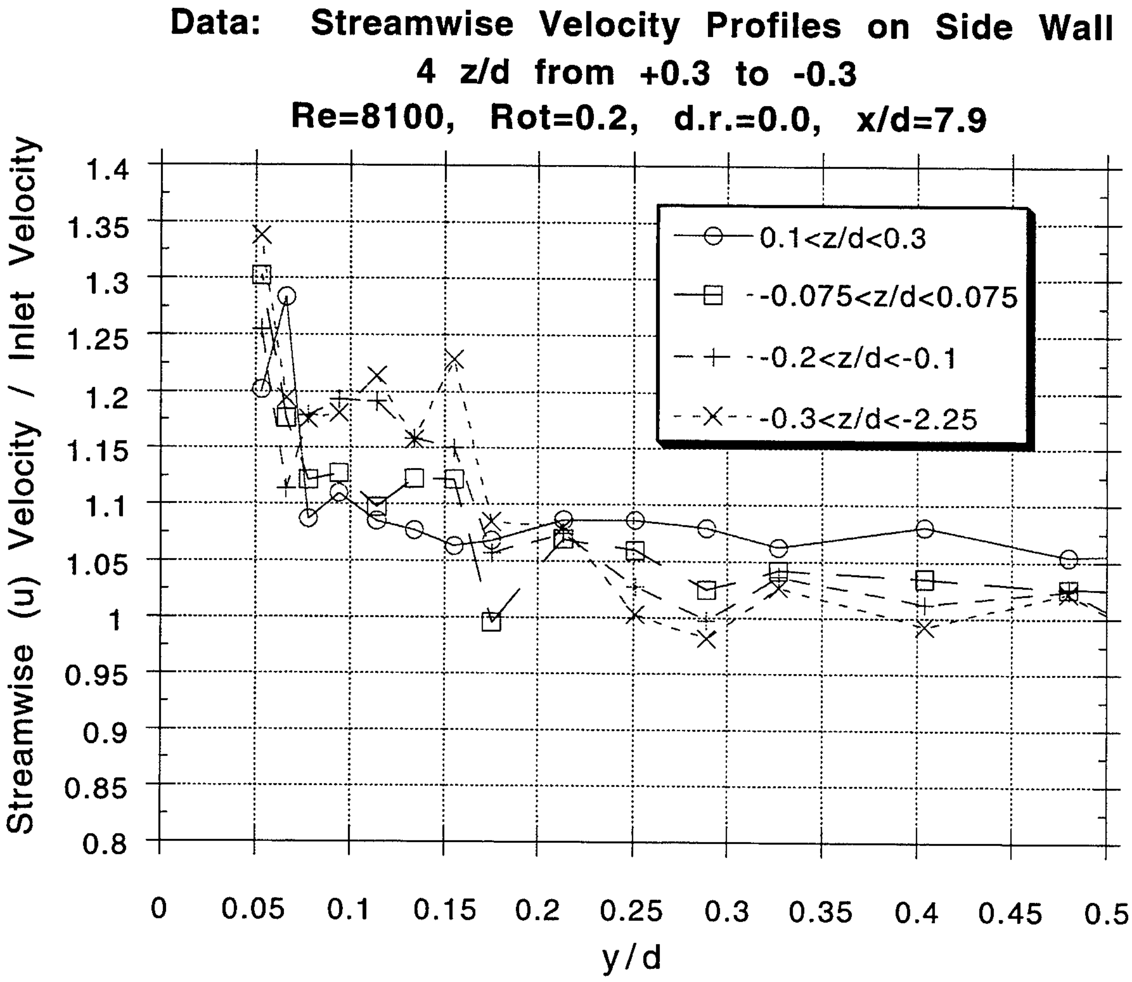


Figure 4.12: Streamwise (y) Velocity Profile Data for Rot=0.2 at 4 z/d Locations

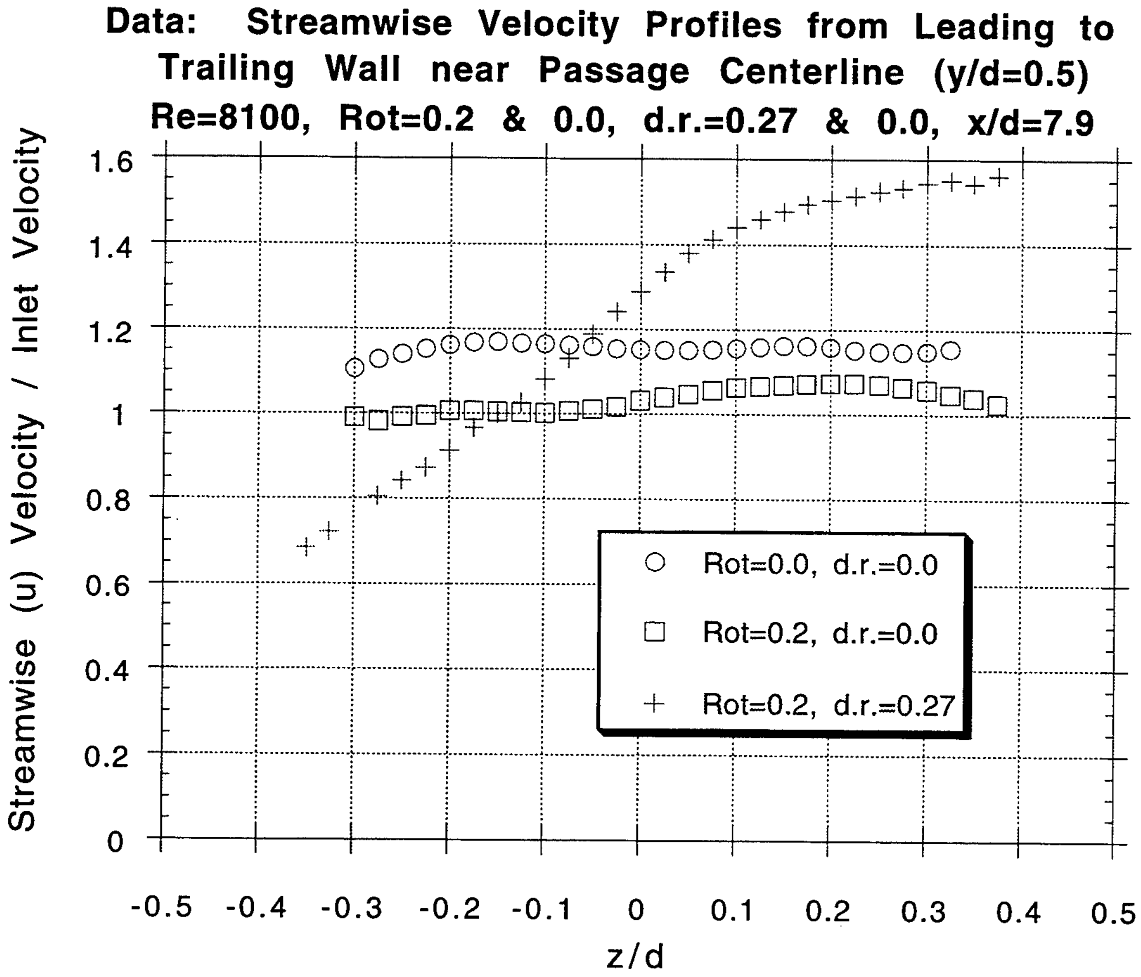


Figure 4.13: Streamwise (z) Velocity Profile Data for 3 Cases: Rot=0 & d.r.=0, Rot=0.2 & d.r.=0, and Rot=0.2 & d.r.=0.27 all at x/d=7.9

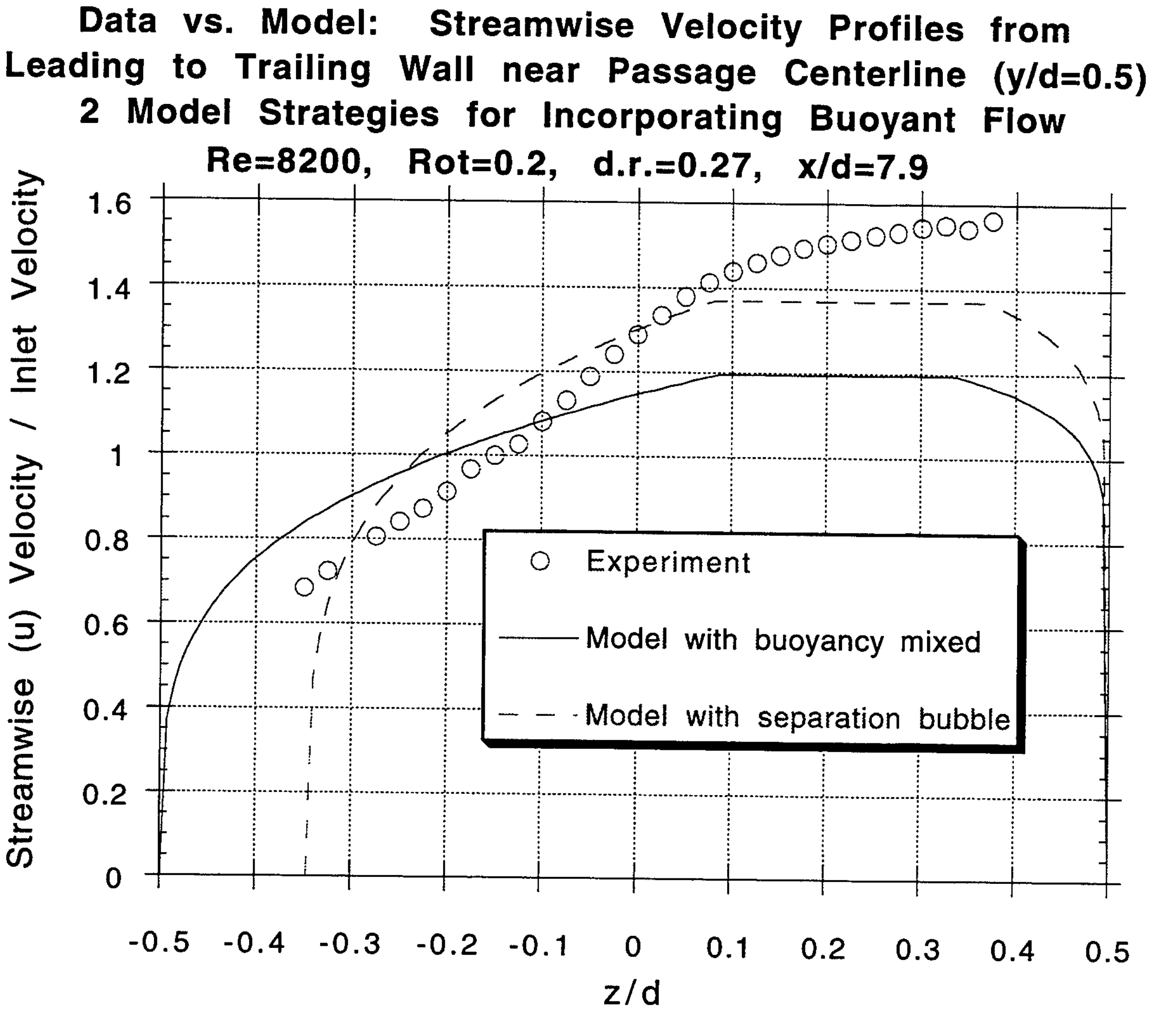
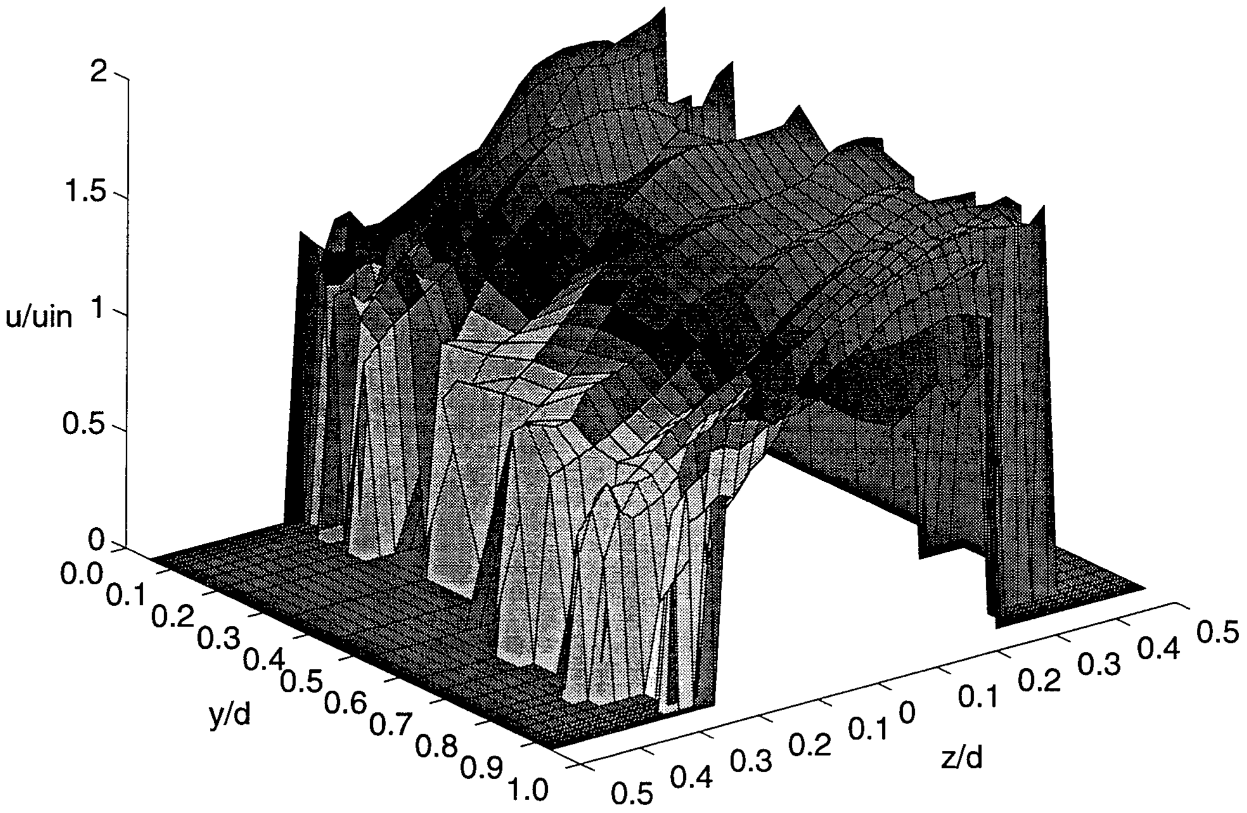
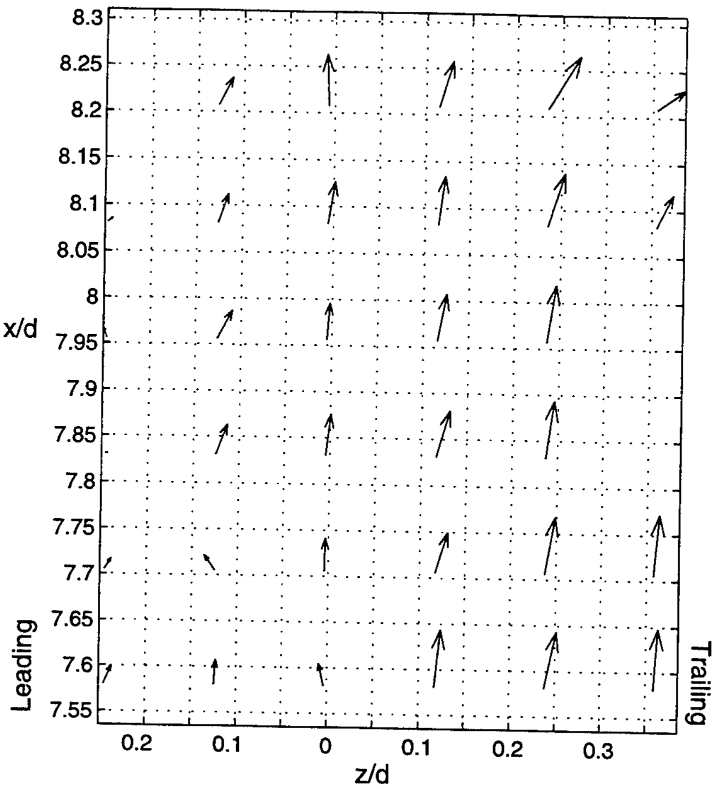


Figure 4.14: Data vs. Model Prediction of Streamwise (z) Velocity Profiles for Rot=0.2 & d.r.=0.27. Effect of Different Model Strategies for Buoyant Flow Mixing.



Re = 8200, Rot = 0.2, & d.r. = 0.27 at x/d = 7.95

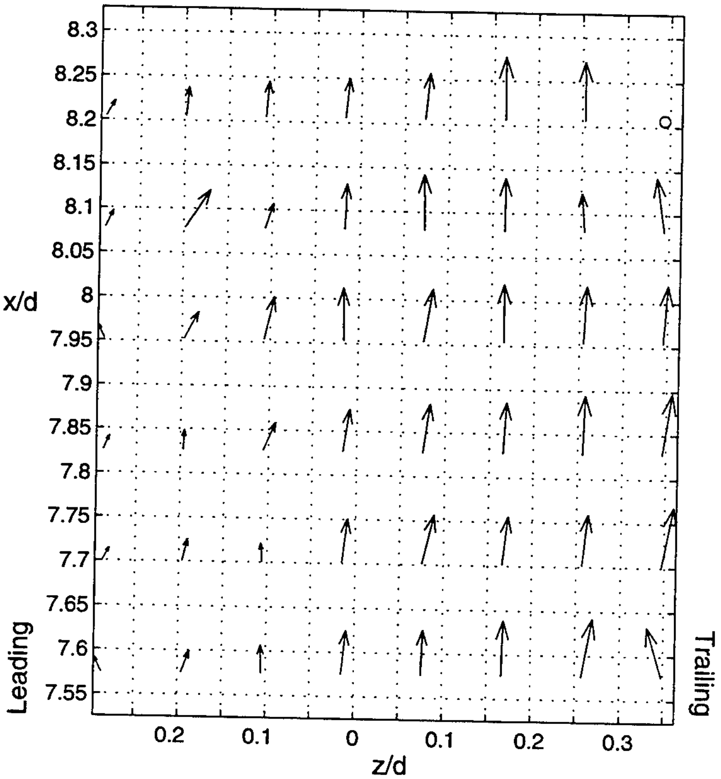
Figure 4.15: Surface Map of Streamwise Velocity Data for Re=8200, Rot=0.2, and d.r.=0.27 at x/d=7.95



Velocity Vectors for
Re=8200
Rot=0.2
d.r.=0.27
at y/d=0.85

mean u/uin=1.06

mean w/uin=+0.24

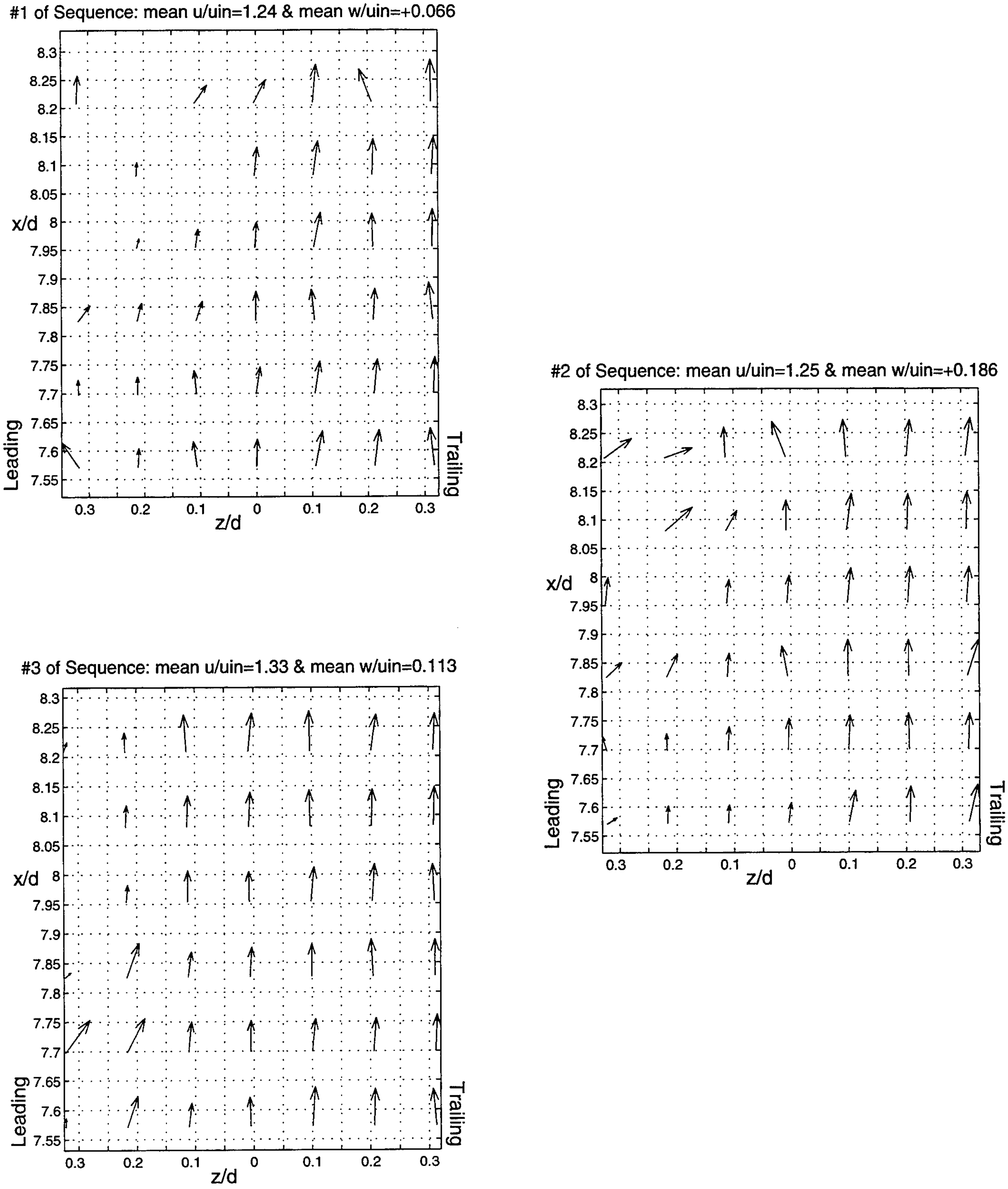


Velocity Vectors for
Re=8200
Rot=0.2
d.r.=0.27
at y/d=0.79

mean u/uin=1.20

mean w/uin=+0.12

Figure 4.16: Sample Vector x-z Plots for Re=8200, Rot=0.2, and d.r.=0.27 at x/d=7.9


#1 of Sequence: mean u/uin=1.24 & mean w/uin=+0.066

#2 of Sequence: mean u/uin=1.25 & mean w/uin=+0.186

#3 of Sequence: mean u/uin=1.33 & mean w/uin=0.113

Figure 4.17: 3 Sequential Vector x-z Plots for Re=8100, Rot=0.2, & d.r.=0.27 at y/d=0.67.

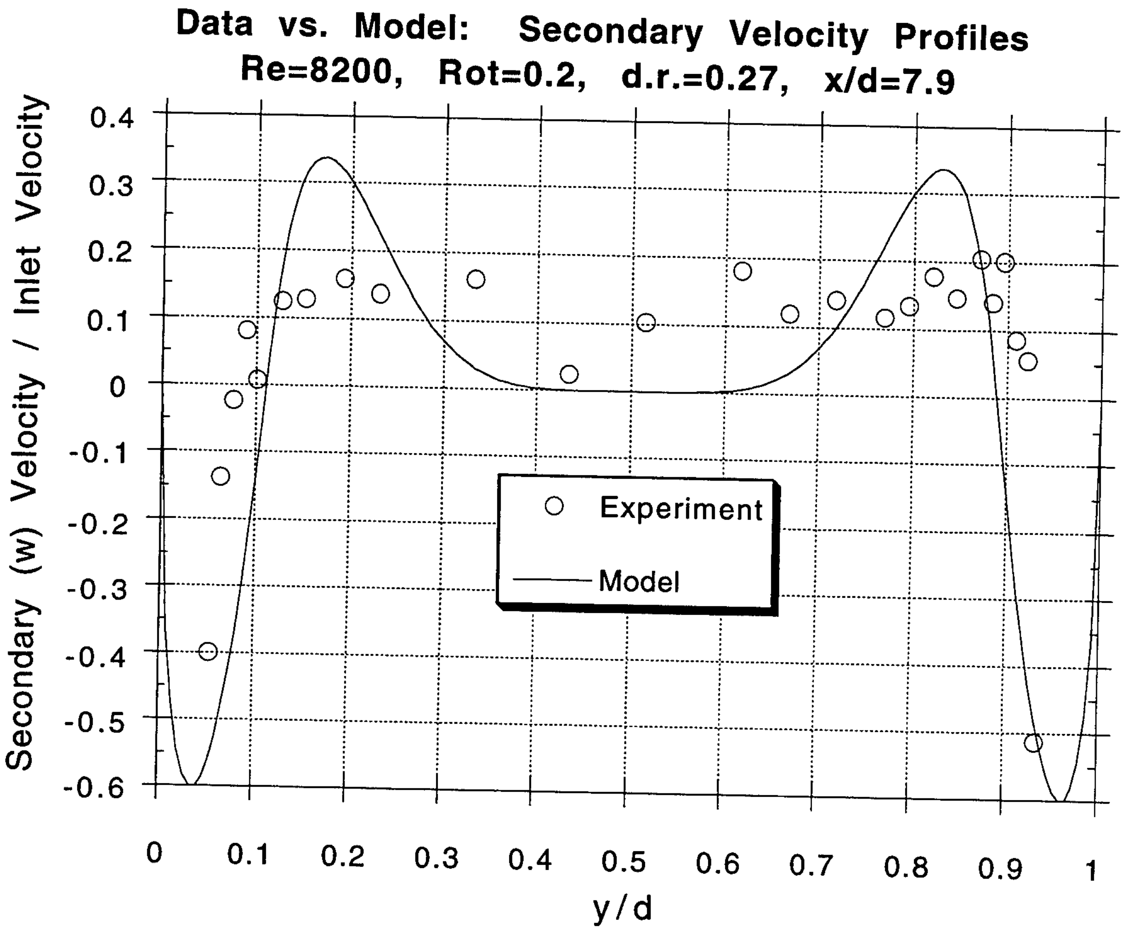


Figure 4.18: Data vs. Model Prediction of Secondary Velocity Profiles for Re=8200, Rot=0.2, and d.r.=0.27 at x/d=7.9

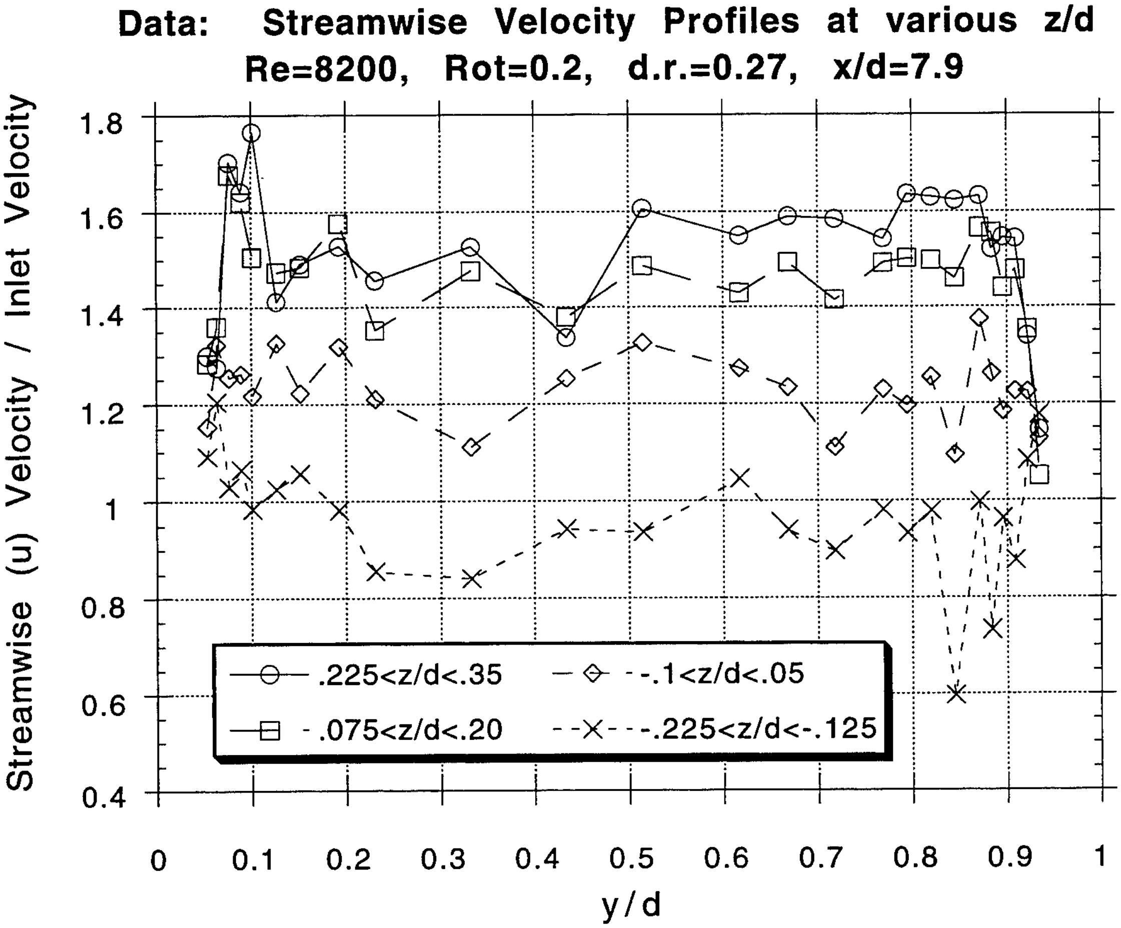


Figure 4.19: Streamwise (y) Velocity Profile Data for Re=8200, Rot=0.2, & d.r.=0.27 at 4 z/d Locations and x/d=7.9

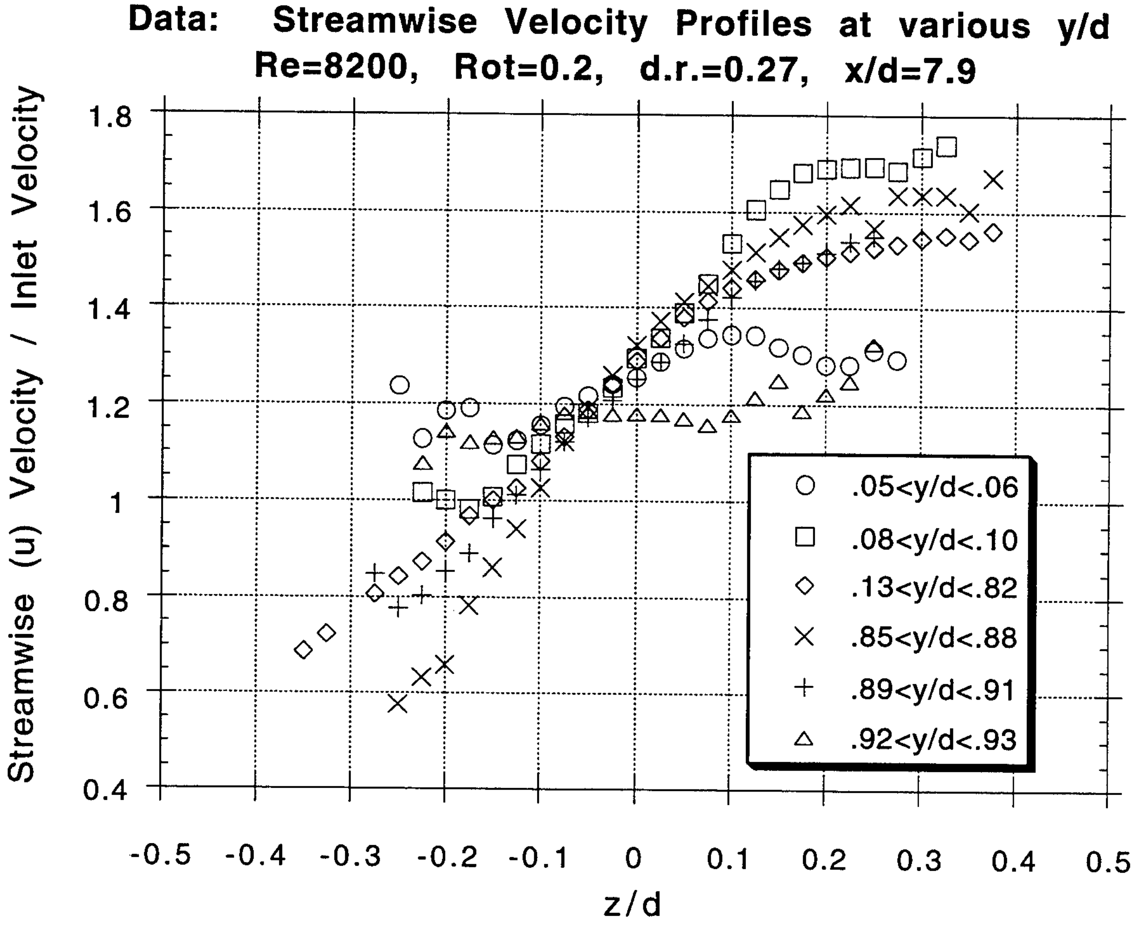


Figure 4.20: Streamwise (z) Velocity Profile Data for Re=8200, Rot=0.2, & d.r.=0.27 at 6 y/d Locations and x/d=7.9

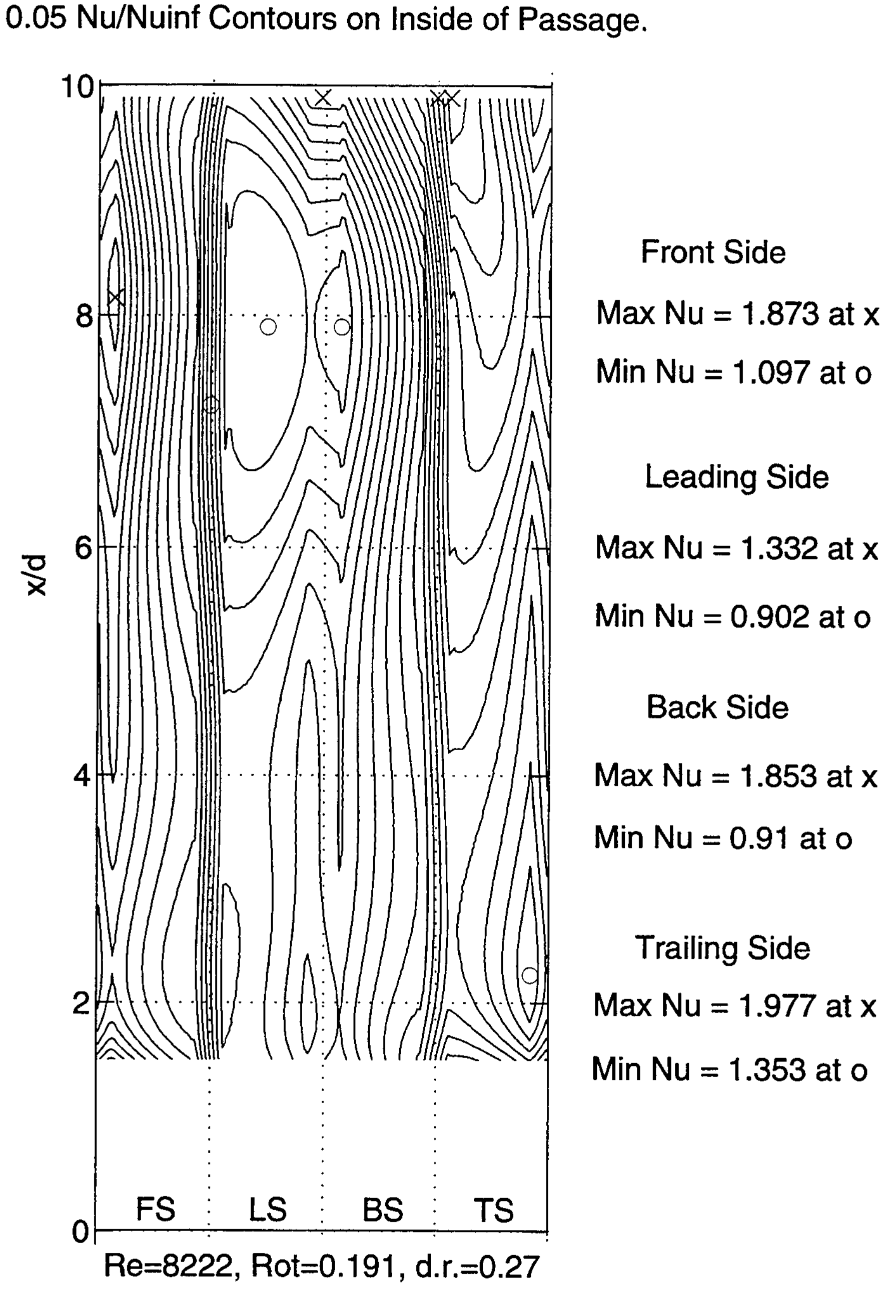


Figure 4.21: Passage Nu Contours for Re=8222, Rot=0.191, & d.r.=0.27.

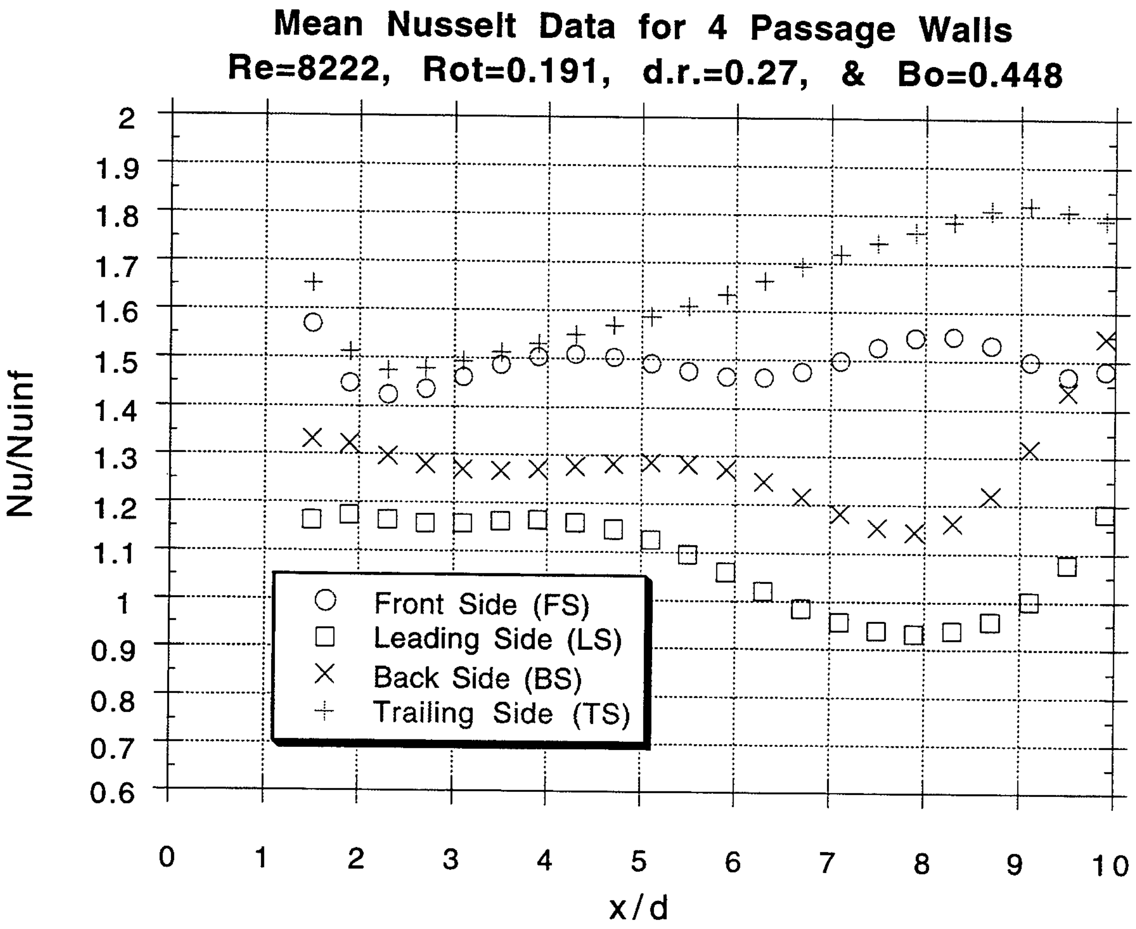


Figure 4.22: Mean Side Nu(x) for all 4 Walls. Re=8222, Rot=0.191, & d.r.=0.27.

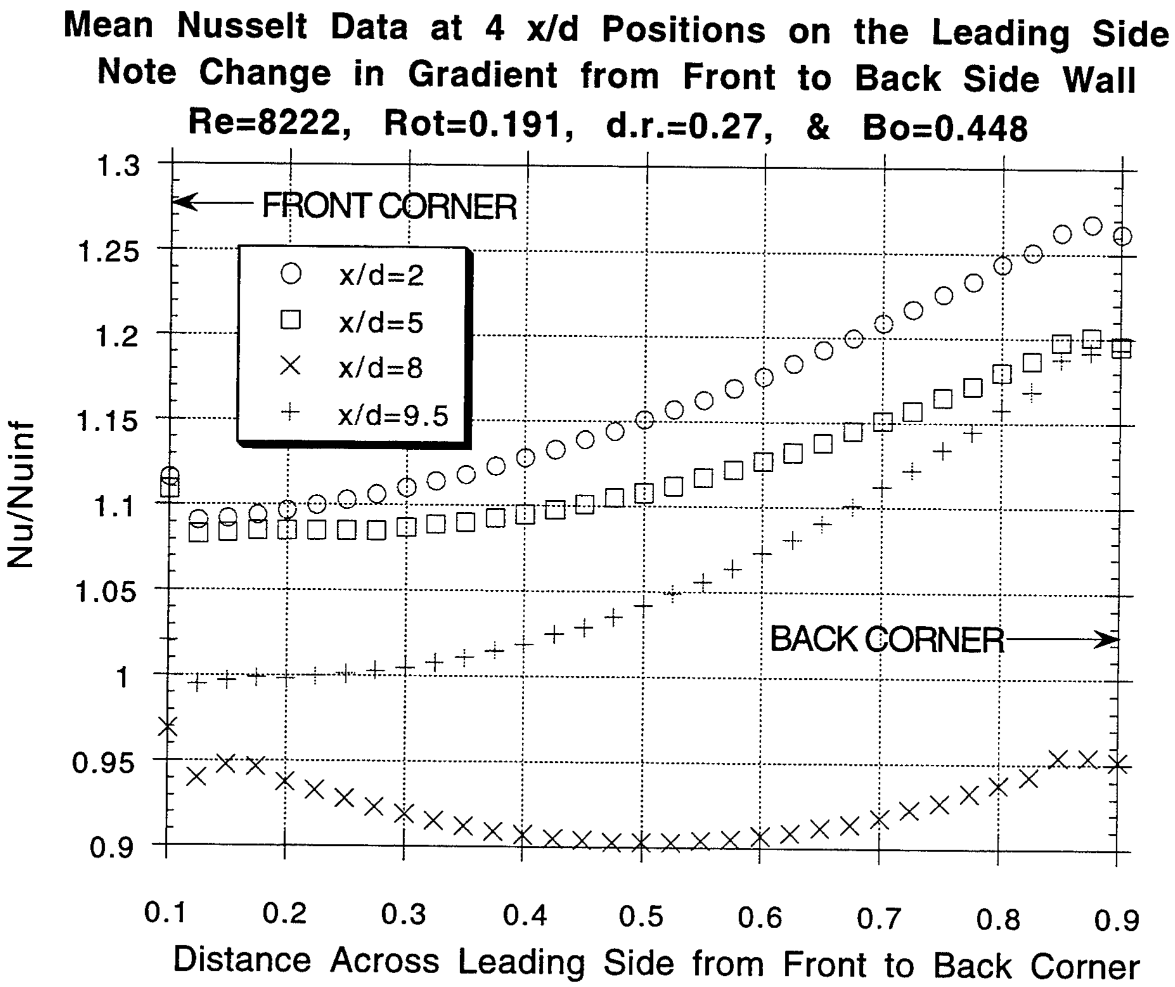


Figure 4.23: Nu Distribution along Leading Wall at 4 x/d Stations.
Re=8222, Rot=0.191, & d.r.=0.27.

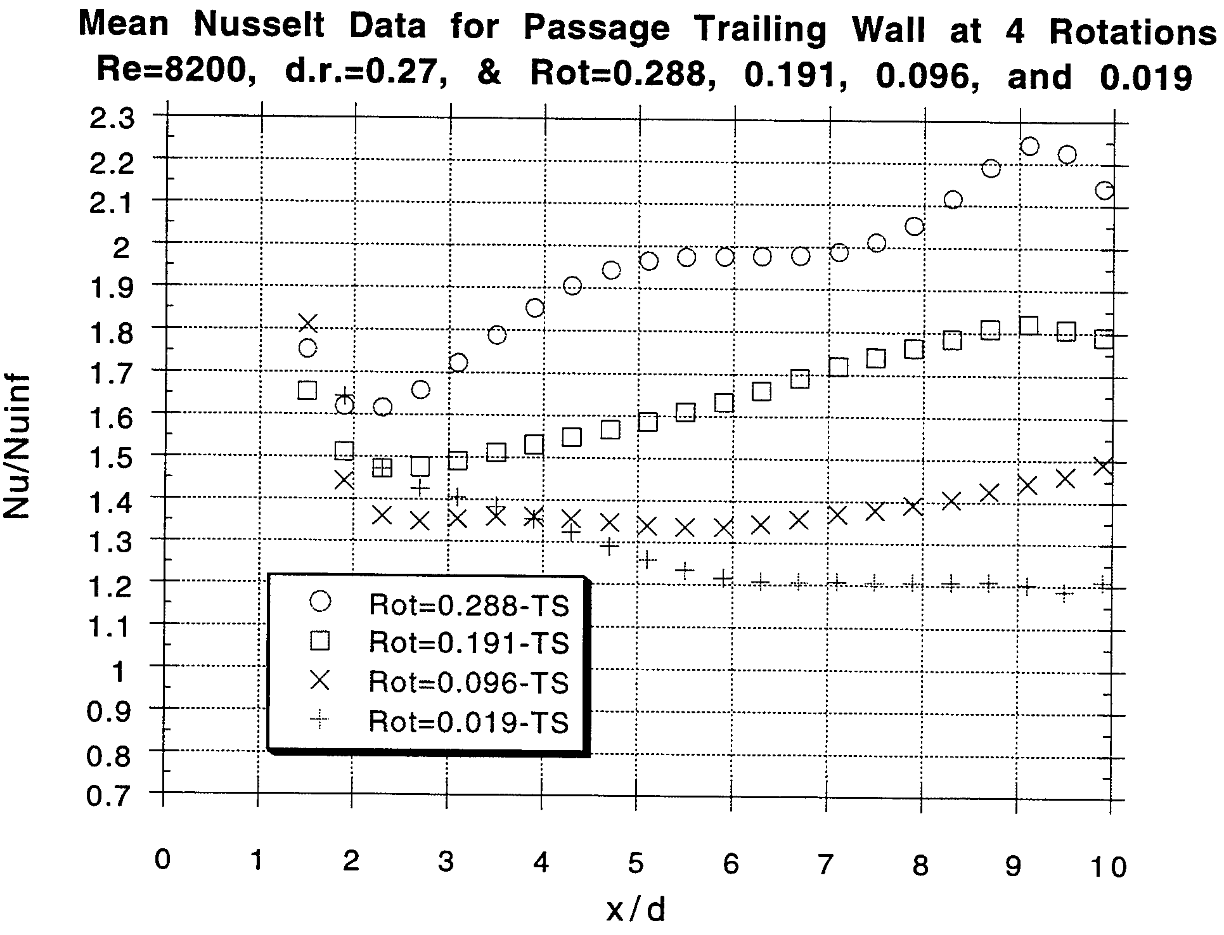


Figure 4.24: Mean Side Nu(x) on Trailing Wall for 4 Rot: 0.288, 0.191, 0.096, & 0.019.

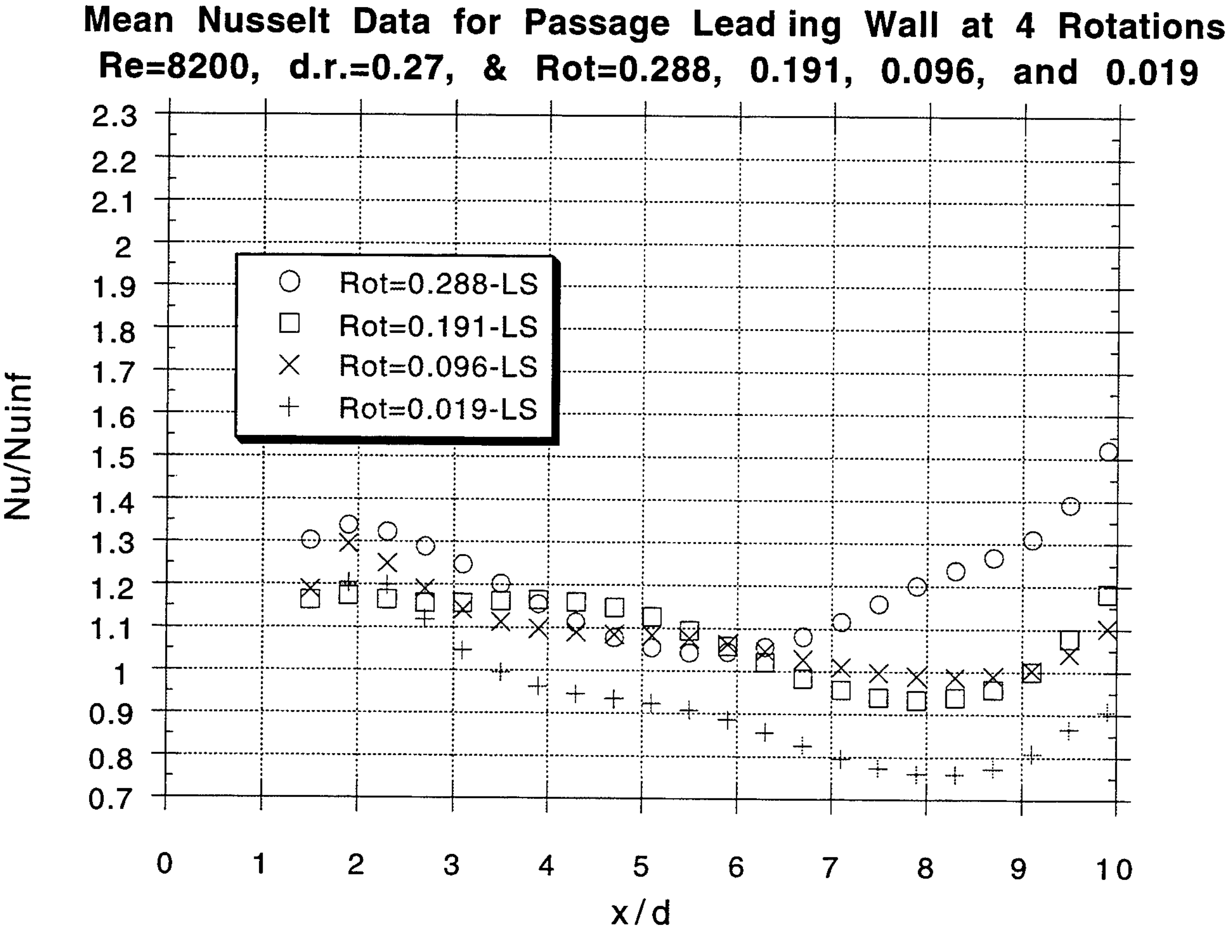


Figure 4.25: Mean Side Nu(x) on Leading Wall for 4 Rot: 0.288, 0.191, 0.096, & 0.019.

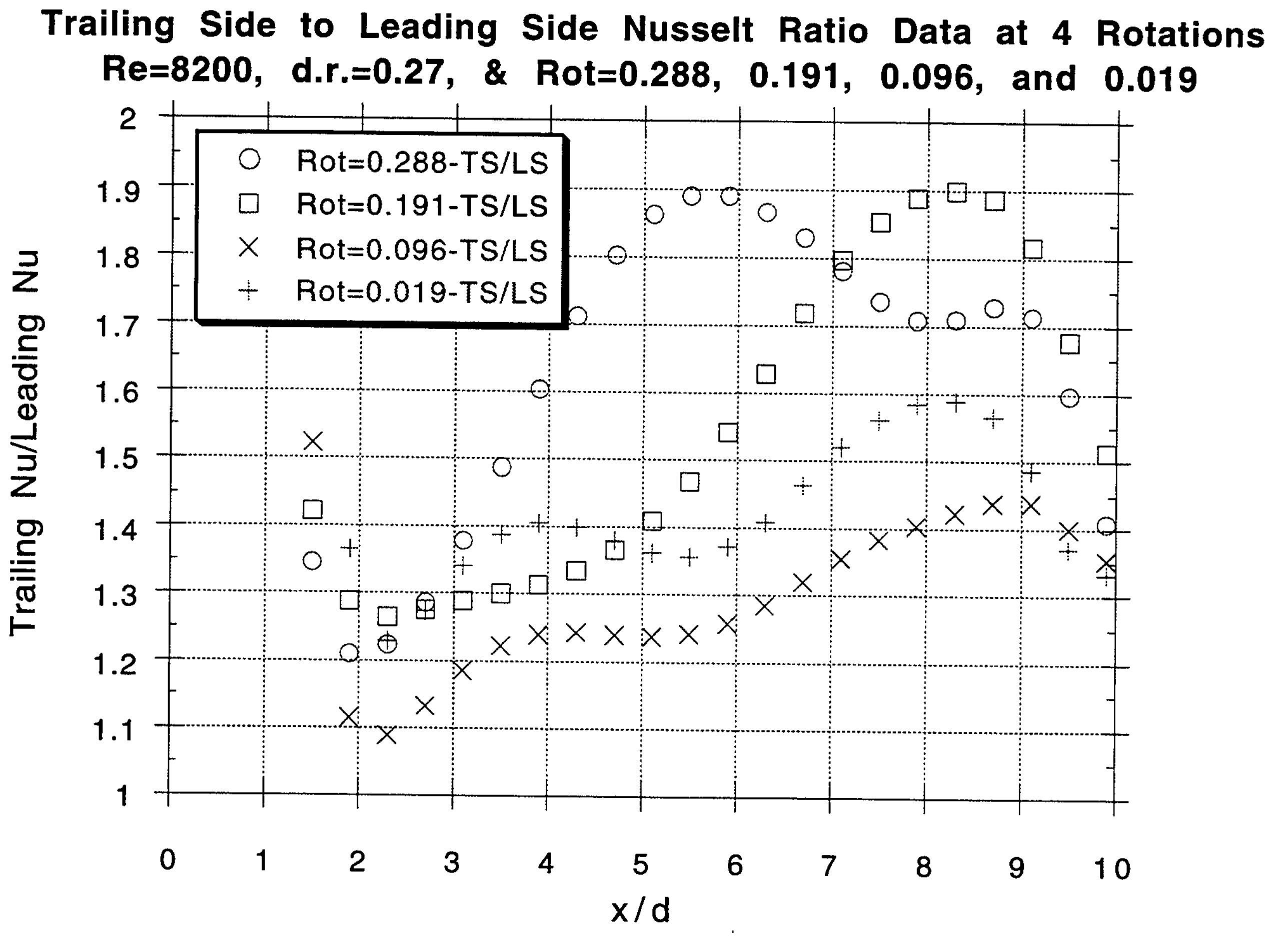


Figure 4.26: Mean Side Trailing Wall Nu to Leading Wall Nu Ratio
for 4 Rot: 0.288, 0.191, 0.096, & 0.019.

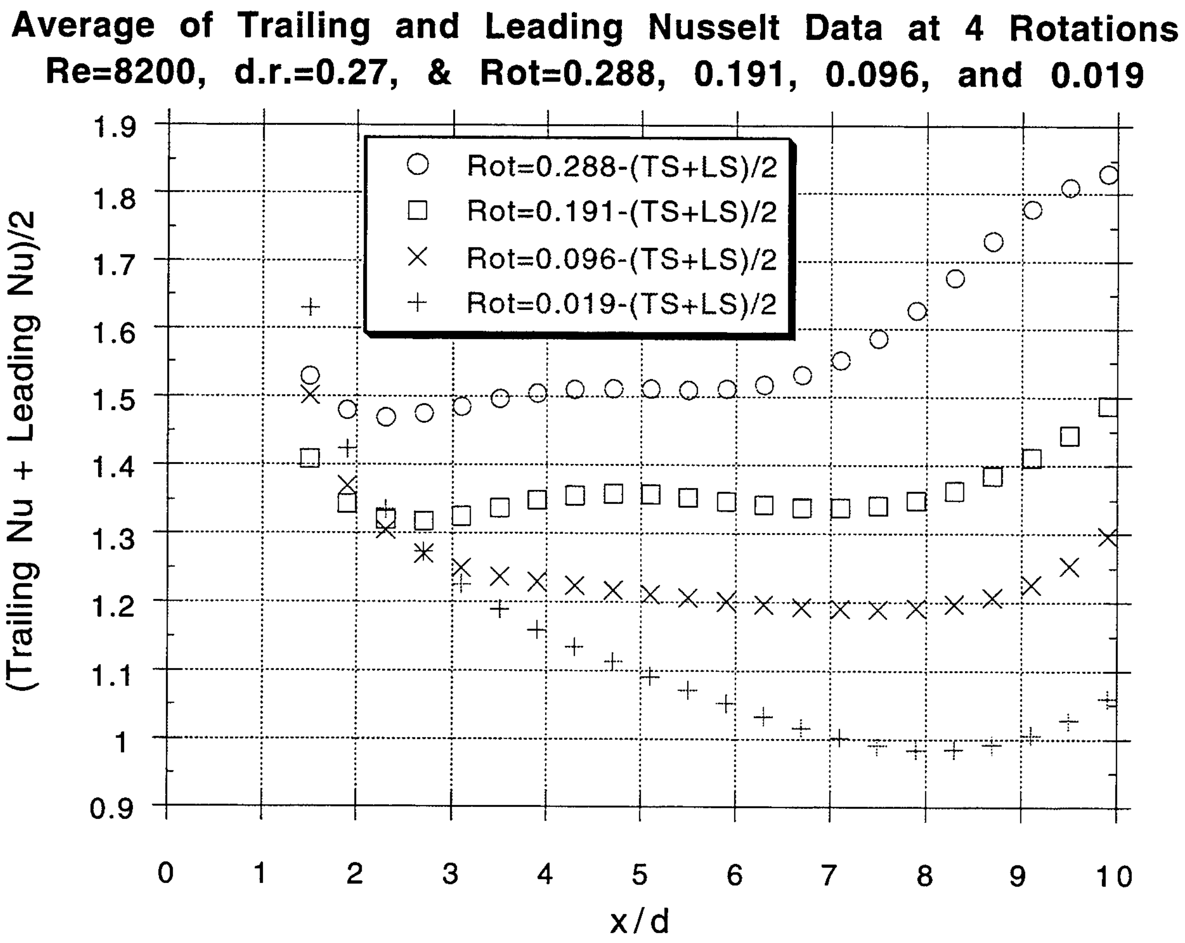


Figure 4.27: Mean Side Trailing Wall Nu and Leading Wall Nu Average
for 4 Rot: 0.288, 0.191, 0.096, & 0.019.

Chapter 5: Summary and Conclusions

Summary

An experimental investigation was conducted on the internal flowfield of a simulated turbine blade cooling passage. The passage is of a square cross-section and was manufactured from quartz for optical accessibility. Velocity measurements were taken using Particle Image Velocimetry for both heated and non-heated cases. Thin film resistive heaters on all four exterior walls of the passage allow heat to be added to the coolant flow without obstructing laser access. Under the same conditions, an infrared detector with associated optics collected wall temperature data for use in calculating local Nusselt number. The test section was operated with radial outward flow and at values of Reynolds number, Rotation number, and density ratio typical of applications.

Ensemble-averaged velocity data for the non-heated case document the evolution of the Coriolis-induced double vortex. The vortex strength increases with Rot and x/d, while its spatial extent decreases with Rot and increases with x/d. The decrease in spatial extent with Rot is due to the streamwise component of the Coriolis acceleration which creates a considerably thinned side wall boundary layer. The vortex has the effect of increasing the leading side boundary layer thickness at the expense of the trailing and side wall boundary layers. Individual vector maps reveal an unsteady, turbulent flowfield in the cooling passage.

Velocity data for the heated case show a strongly distorted streamwise profile indicative of a buoyancy effect on the leading side. The ratio of edge velocity on the trailing wall to leading wall is approximately 2.2 (vs. 1.1 for the non-heated case). The Coriolis vortex is the mechanism for the accumulation of stagnant flow on the leading side of the passage. The vortex strength and position do not change noticeably with heat addition. The high temperature side wall and trailing wall boundary layer fluid is swept by the vigorous motion of the Coriolis crossflow onto the leading wall. Heat transfer data show a maximum factor of two difference in the Nusselt number from trailing side to leading side at x/d=8. An estimate of this heat transfer disparity based on the measured boundary layer edge velocity yields approximately the same factor of two. A manufacturing anomaly which created a higher heat flux on the front side wall than the back side wall appears to be the cause of a stronger vortex and higher heat transfer on the front side wall. The leading wall Nu data show a corresponding depression in Nu near the front side wall. This depression appears to move across the leading wall with x/d, indicative of bulk steady motion of the core flow.

A momentum integral model was developed for data interpretation. The model assumes the flow can be adequately characterized by an inviscid core flow surrounded by wall shear layers. Calculated streamwise profiles and secondary flows match the experimental data well. The model, the velocity data, and the heat transfer data suggest the presence of separated flow on the leading wall starting at about five passage widths for the conditions studied. Nu data suggest that the extent of this separated region moves upstream with higher Rot.

Conclusions

This thesis contains the first reported global velocity measurements in a heated, rotating test section. In addition, high resolution heat transfer measurements taken with the same test section operating at the same conditions allow the direct correlation of flow phenomena with heat transfer phenomena.

The Coriolis vortex has two significant effects on the flowfield. It transports hot, low momentum wall fluid from the trailing wall to the side wall and finally to the leading wall of the square passage. Second, it creates a rise in streamwise velocity in the region near the side wall where the secondary flow is in the direction of rotation. The first of these two effects has the greatest impact on cooling. The accumulation of low-momentum coolant near the leading wall effectively deters heat removal here.

The effect of buoyancy is also considerable, reducing streamwise velocities by a factor of 2 in regions where high temperature coolant is concentrated (near the leading wall).

A factor of 2 difference in Nu from trailing to leading wall arises due to the combination of these buoyancy and Coriolis effects. Without the Coriolis vortex, the buoyancy effect would be evenly distributed to all 4 walls. Without the buoyancy effect, the Coriolis vortex would only create a 10% shift in streamwise velocity toward the trailing wall (vs. 120% with buoyancy). Since the wall Nu correlates directly with this edge velocity, a Nu prediction based solely on the Coriolis effect would have a factor of 2 error. This underscores the need for full simulation of both effects simultaneously. Measuring each effect separately and then assessing their mutual impact on heat transfer by superposition (as done by Tse et al [48,49]) is inaccurate.

Individual vector fields at a fixed x/d location exhibit flow unsteadiness and suggest lateral motion (along the y axis) of the Coriolis vortex with time. The effect of this motion on steady state heat transfer measurements is undetermined. Heat transfer data along the leading and trailing walls suggest a steady lateral migration of the bulk flow with x/d. The source of this lateral motion is postulated to be the growth and collapse of a separation bubble on the leading wall. These represent areas of further research.

A momentum integral model has proven to be a useful tool in flow interpretation. With several refinements incorporated from the experimental data, the model calculations are accurate to within 20% of the observed flow features. This lends validity to the underlying model assumption of an inviscid core flow surrounded by independent wall shear layers. Modeling the Coriolis vortex as a momentum deficit transport mechanism that primarily exchanges hot, low-momentum boundary layer fluid between adjacent walls was found to be a successful strategy. Modeling the buoyancy effect as a separated region of stagnant flow on the leading wall also matches the experimental observation well.

Finally, Particle Image Velocimetry (PIV) has been successfully applied to a rotating flowfield. A novel diagnostic technique is employed to remove the test section rotation from the resultant double image. The use of thin film heaters was also validated for providing a uniform surface heat flux without blocking optical access to the flowfield.

Future Work

With the velocity measurement technique now proven, several avenues of further research are available. The walls of the test section used in this study do not have turbulence promoters (ribs), however nothing should preclude taking the same data set with a ribbed passage. Also, the effects of flow direction, Reynolds number, passage aspect ratio, and inlet conditions can be readily investigated. By tailoring the optical system appropriately, the near wall regions could be interrogated for reverse flow on the leading wall and the second set of vortices predicted by many CFD calculations. Finally, a CFD calculation using these exact experimental conditions would allow direct comparison with a measured velocity field. This would contribute to code validation efforts.

REFERENCE LIST

[1] Adrian, R., 1991, "Particle-Imaging Techniques for Experimental Fluid Mechanics", Annual Review of Fluid Mechanics, Vol 23, pp. 261-304.

[2] Barry, P., 1994, "Rotational Effects on Turbine Blade Cooling", Master's Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA.

[3] Barua, S.N., 1954, "Secondary Flow in a Rotating Straight Pipe", Proc. Royal Soc. A, Vol. 227, pp. 133-139.

[4] Berg, H., Hennecke, D., Elfert, M., and Hein, O., 1991, "The Effect of Rotation on Local Coolant Side Flow and Heat Transfer in Turbine Blades", ISABE 91-7016, published by AIAA, pp. 170-183.

[5] Blair, M., 1983, "Influence of Free-Stream Turbulence on Turbulent Boundary Layer Heat Transfer and Mean Profile Development, Part I - Experimental Data", ASME Journal of Heat Transfer, Vol. 105, pp. 33-47.

[6] Bonhoff, B., Tomm, U., Johnson, B., and Jennions, I., 1997, "Heat Transfer Predictions for Rotating U-Shaped Coolant Channels with Skewed Ribs and with Smooth Walls", ASME Paper #97-GT-162, presented at ASME IGTI, Orlando, Florida, June 2-5.

[7] Boussinesq, 1930, "Theorie Analytique de la Chaleur", Vol 2, Gathiers-Villars, Paris.

[8] Boyer, D.L., 1965, "Flow through a Rapidly Rotating Rectangular Channel", PhD Thesis, The Johns Hopkins University.

[9] Bryanston-Cross, P. and Epstein, A., 1990, "The Application of Sub-Micron Particle Visualisation for PIV at Transonic and Supersonic Speeds", Prog. Aerospace Sci., Vol. 27, pp. 237-265.

[10] Chew, J., 1993, "A Momentum-Integral Solution of Flow in a Rotating Circular Duct", Int. J. of Heat and Fluid Flow, Vol. 14, No. 3, pp. 240-245.

[11] Dittus, F.W., and Boelter, L.M.K., 1930, University of California Publications, Vol 2, pg. 443.

[12] Dutta, S., Han, J.C., and Lee, C., 1994, "Local Heat Transfer in a Rotating Two-Pass Triangular Duct with Smooth Walls", ASME paper #94-GT-337, presented at ASME IGTI, The Hague, Netherlands, June 13-16.

[13] Dutta, S., Andrews, M., and Han, J.C., 1994, "Numerical Prediction of Turbulent Heat Transfer in a Rotating Square Duct with Variable Rotational Buoyancy Effects", presented at the 6th AIAA/ASME Thermophysics and Heat Transfer Conference, Colorado Springs, CO, June 20-23.

[14] El-Husayni, H., Taslim, M., and Kercher, D., 1994, "Experimental Heat Transfer Investigation of Stationary and Orthogonally Rotating Asymmetric and Symmetric

Heated Smooth and Turbulated Channels", Journal of Turbomachinery, Vol. 116, pp. 124-132.

[15] Govatzidakis, G., 1995, "Heat Transfer in Rotating Passages", Master's Thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA.

[16] Guidez, J., 1989, "Study of the Convective Heat Transfer in a Rotating Coolant Channel", ASME Journal of Turbomachinery, Vol. 111, pp. 43-50.

[17] Han, J. and Zhang, Y., 1992, "Effect of Uneven Wall Temperature on Local Heat Transfer in a Rotating Square Channel with Smooth Walls and Radial Outward Flow", ASME Journal of Heatr Transfer, Vol. 114, pp. 850-858.

[18] Han, J., Zhang, Y., and Lee, C., 1994, "Influence of Surface Heating Condition on Local Heat Transfer in a Rotating Square Channel with Smooth Walls and Radial Outward Flow", ASME Journal of Turbomachinery, Vol. 116, pp. 149-158.

[19] Hancock, P. and Bradshaw, P., 1983, "The Effect of Free-Stream Turbulence on Turbulent Boundary Layers", Journal of Fluids Engineering, Vol. 105, pp 284-289.

[20] Harasgama, S. and Morris, W., 1988, "The Influence of Rotation on the Heat Transfer Characteristics of Circular, Triangular, and Square-Sectioned Coolant Passages of Gas Turbine Rotor Blades", ASME Journal of Turbomachinery, Vol. 110, pp. 44-50.

[21] Hill, P. and Moon, I., 1962, "Effects of Coriolis on the Turbulent Boundary Layer in Rotating Fluid Mechanics", Gas Turbine Lab Report #69, Massachusetts Institute of Technology, Cambridge, MA.

[22] Howard, J., Patankar, S., and Bordynuik, R.,1980, "Flow Prediction in Rotating Ducts Using Coriolis-Modified Turbulence Models", ASME Journal of Fluids Engineering, Vol. 102, pp. 456-461.

[23] Hsieh, S., Chiang, M., Chen, P., 1997, "Velocity Measurements and Local Heat Transfer in a Rotating Ribbed Two-Pass Square Channel with Uneven Wall Heat Flux", ASME Paper #97-GT-160, presented at ASME IGTI, Orlando, Florida, June 2-5.

[24] Iacovides, H. and Launder, B., 1991, "Parametric and Numerical Study of Fully Developed Flow and Heat Transfer in Rotating Rectangular Ducts", ASME Journal of Turbomachinery, Vol. 113, pp. 331-338.

[25] Ito, H. and Nanbu, K., 1971, "Flow in Rotating Straight Pipes of Circular Cross Section", ASME Journal of Basic Engineering, pp. 383-394.

[26] Jefferies, R.W., 1996, "Interactions of a Quasi-Two-Dimensional Vortex with a Stationary and Oscillating Leading-Edge", PhD Thesis, Department of Mechanical Engineering, Lehigh University, Pa.

[27] Johnson, B., Wagner, J., Steuber, G., and Yeh, F., 1994, "Heat Transfer in Rotating Serpentine Passages with Trips Skewed to the Flow", ASME Journal of Turbomachinery, Vol. 116, pp. 113-123.

[28] Johnson, B., Wagner, J., Steuber, G., and Yeh, F., 1994, "Heat Transfer in Rotating Serpentine Passages with Selected Model Orientations for Smooth or Skewed Trip Walls", ASME Journal of Turbomachinery, Vol. 116, pp. 738-744.

[29] Jones, A., 1994, "The Effects of Rotation on Heat Transfer in Turbine Blade Cooling", Master's Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA.

[30] Kays, W. and Crawford, M., *"Convective Heat and Mass Transfer"*, McGraw-Hill Book Company, 1980.

[31] Kheshgi, H. and Scriven, L., 1985, "Viscous Flow through a Rotating Square Channel", Phys. Fluids, Vol 28, pp. 2968-2979.

[32] Khodak, A., Kirillov, A., Ris, V., and Smirnov, E., 1993?, "Local Heat Transfer in 3-D Turbulent Flow through Ducts Rotating in the Orthogonal Mode", 8-IC-14, pp. 261-266.

[33] Kline, S.J., and McClintock, F.S., 1953, "Describing Uncertainties in Single-Sample Experiments", Mechanical Engineering, Jan., pp. 3-8.

[34] Kreatsoulas, J.C., 1983, "An Experimental Study of Impingement Cooling in Rotating Turbine Blades", Gas Turbine Lab Report #178, Massachusetts Institute of Technology, Cambridge, MA.

[35] Kuo, C. and Hwang, G., 1994, "Aspect Ratio Effect on Convective Heat Transfer of Radially Outward Flow in Rotating Rectangular Ducts", Int'l Journal of Rotating Machinery, Vol. 1, No., 1, pp. 1-18.

[36] Medwell, J., Morris, W., Xia, J., and Taylor, C., 1991, "An Investigation of Convective Heat Transfer in a Rotating Coolant Channel", ASME Journal of Turbomachinery, Vol. 113, pp. 354-359.

[37] Melling, A., 1986, "Seeding Gas Flows for Laser Anemometry", AGARD Conference, Advanced Instrumentation for Aero Engine Components, Philadelphia, 19-23 May, paper #15, AGARD CP 399.

[38] Mills, A., *"Heat Transfer"*, Richard D. Irwin Inc., 1992.

[39] Moore, J., 1967, "Effects of Coriolis on Turbulent Flow in Rotating Rectangular Channels", Gas Turbine Lab Report #89, Massachusetts Institute of Technology, Cambridge, MA.

[40] Mori, Y. and Nakayama, W., 1968, "Convective Heat Transfer in Rotating Radial Circular Pipes (1st Report, Laminar Region)", International Journal of Heat and Mass Transfer, Vol. 11, pp. 1027-1040.

[41] Morris, W. and Ayhan, T., 1979, "Observations on the Influence of Rotation on Heat Transfer in the Coolant Channels of Gas Turbine Rotor Blades", Proc. Insts. Mech Engrs, Vol 193, pp. 303-311.

[42] Morris, W., 1981, Heat Transfer and Fluid Flow in Rotating Coolant Channels, Research Studies Press.

[43] Morris, W. and Ghavami-Nasr, G., 1991, "Heat Transfer Measurements in Rectangular Channels with Orthogonal Mode Rotation", ASME Journal of Turbomachinery, Vol. 113, pp. 339-345

[44] Potter, M. and Foss, J., "*Fluid Mechanics*" , Great Lakes Press, Inc., 1982.

[45] Prakash, C. and Zerkle, R., 1992, "Prediction of Turbulent Flow and Heat Transfer in a Radially Rotating Square Duct", ASME Journal of Turbomachinery, Vol. 114, pp. 835-846.

[46] Tekriwal, P., 1996, "Effect of Aspect Ratio on the Buoyancy Driven Reverse Flow near the Leading Wall of Rotating Cooling Passages", ASME Paper #96-GT-173, presented at ASME IGTI, Birmingham, UK, June 10-13.

[47] Tekriwal, P., 1997, "Heat Transfer Predictions in Rotating Radial Smooth Channel: Comparative Study of k-e Models with Wall Function and Low-Re Model", ASME Paper #97-GT-162, presented at ASME IGTI, The Hague, Netherlands, June 13-16.

[48] Tse, D. and McGrath, D., 1995, "A Combined Experimental/Computational Study of Flow in Turbine Blade Passage: Part I - Experimental Study", ASME Paper #95-GT-355, presented at ASME IGTI, Houston, Texas, June.

[49] Tse, D. and Steuber, G., 1997, "Fow in a Rotating Square Serpentine Coolant Passage with Skewed Trips", ASME Paper #97-GT-529, presented at ASME IGTI, Orlando, Florida, June 2-5.

[50] Uellner, S. and Roesner, K., 1991, "LSV Applied to the Flow in a Rotating Channel", ASME Laser Anemometry, Vol. 2, pp. 541-546.

[51] Wagner, J., Johnson, B., and Hajek, T., 1991, "Heat Transfer in Rotating Passages with Smooth Walls and Radial Outward Flow", ASME Journal of Turbomachinery, Vol. 113, pp. 42-51.

[52] Wagner, J., Johnson, B., and Kopper, F., 1991, "Heat Transfer in Rotating Serpentine Passages with Smooth Walls", ASME Journal of Turbomachinery, Vol. 113, pp. 321-330.

[53] Wagner, J., Johnson, B., Graziani, R., and Yeh, F., 1991, "Heat Transfer in Rotating Serpentine Passages with Trips Normal to Flow", ASME paper #91-GT-260, presented at ASME IGTI, Orlando, Florida, June 3-6.

[54] Wagner, R. and Velkoff, H., 1972, "Measurements of Secondary Flows in a Rotating Duct", ASME Journal of Engineering for Power, pp. 261-270.

APPENDIX A: MOMENTUM INTEGRAL MODEL CODE

```
/* */
/*      Determine the effect of the Coriolus-induced counter-rotating
        vortices and Buoyancy due to Temperature gradients on the
        flow in our test section
                   jpbcoby7.c updated 5 Nov 1996 for air
                   jpbcoby8.c updated 19 Nov 1996 for secondary
                        flow profiles which are not parabolic
                   jpbcoby9.c updated for new vsidorig
                   jpbcob11.c many new updates on 13 May 1997
                   jpbcob13.c change print output for Matlab
                   jpbcob20.c 1/9th profiles 16 May 1997
                   jpbcob21.c more updates      16 May 1997
                   jpbcob22.c more updates       19 May 1997
                   jpbcob23.c add the u coriolis effect  21 May 1997
                   jpbcob24.c modifies the u coriolis implementation 27 May 97
                   jpbcob25.c cleanup of jpbcob.24            27 May 97
                   jpbcob26.c reworks buoyancy implementation  27 May 97
                   jpbcob27.c alternate buoyancy implementation  28 May 97
                   jpbcob28.c allow for variable Twall           30 May 97
                   jpbcob29.c calculate temperature profiles based on
                               conservation of energy            31 May 97
                   jpbcob30.c apply Boussinesq to energy conservation
                               integrals by using densin uniformly
                   jpbcob31.c add separated flow region on leading wall 5 Jun 97
                   jpbcob32.c add the w buoyancy effect                  9 Jun 97*/

/* */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define Max_j  1501
#define Max_n  401

int     J,j,N,n,r,raxit,ans,ansB,interv,remaind,Beg,End,thermalbc,nstep,g,jsep;

float   Redh, Rot, dh, dens, visc, kvisc, uin, wrad, wrps, dt[Max_n],
        uls[Max_n][Max_j], usid[Max_n][Max_j], uts[Max_n][Max_j],
        wls[Max_n][Max_j], vsid[Max_n][Max_j], wts[Max_n][Max_j],
        wlsmax[Max_n], vsidmax[Max_n], wtsmax[Max_n],
        dubuoyls[Max_j], dubuoysid[Max_j], dubuoyts[Max_j],
        vsidorig[Max_n][Max_j],
        Tls[Max_n][Max_j], Tsid[Max_n][Max_j], Tts[Max_n][Max_j],
        Twls[Max_n], Twsid[Max_n], Twts[Max_n], Twavg, Tavg, qest,
        denslsmni, denstsmni,
        denssidm[Max_j], denslsm[Max_j], denstsm[Max_j], denssidmni, denssidmno,
        densts[Max_n][Max_j], denssid[Max_n][Max_j], densls[Max_n][Max_j],
        ublthk_ls[Max_n], ublthk_sid[Max_n], ublthk_ts[Max_n],
        udispthk_ls[Max_n], udispthk_sid[Max_n], udispthk_ts[Max_n],
        umomthk_ls[Max_n], umomthk_sid[Max_n], umomthk_ts[Max_n],
        Aint_ls, Aint_sid, Aint_ts,
        sumdefls, sumdefts,
        sumdefls_bmix, sumdefsid_bmix, sumdefts_bmix,
        jeye,
        intls, intsid, intts, pls, psw, pts,
        xequiv_ls[Max_n], xequiv_sid[Max_n], xequiv_ts[Max_n],
        jblls, jblsid, jblts, dely, coeff, uinfpwr,
        delx, x[Max_n], vorteye[Max_n], mdotsw, dvsiddyeye, mpow,
        uinftbl[Max_n], npow,
        int_0, int_1, int_2, c4, ratio, specheat,
        Tin, densin, Pin, Rgas, Rin, densrat, amplit, Tuperc, heatflux,
        ublthk_lsmax, ublthk_sidmax, ublthk_tsmax,
        jdiff, ucor[Max_n][Max_j], ducor[Max_j], ducorp[Max_j], sumcorshft,
        ucorp[Max_j], c6, gmin, mdot, mdotrat, qwls, qwsid, qwts,
        Tavgls[Max_n], Tavgsid[Max_n], Tavgts[Max_n],
        uavgls[Max_n], uavgsid[Max_n], uavgts[Max_n],
        davgls[Max_n], davgsid[Max_n], davgts[Max_n],
        nTls, nTsid, nTts, delQlsin[Max_n], delQtsout[Max_n], delQcore[Max_n],
        delQsum1, delQsum2, delQsum3, mbunmix, ysep, percbls,
        dwlsbuoy[Max_j], dwtsbuoy[Max_j], dwlsmn, dwtsmn, deflswbuoy, deftswbuoy;

FILE *fp;
FILE *fq;
FILE *fx;
FILE *fy;
FILE *fi;
```

```
int energcons(void);
float lsturbbl(float pls);
float sidturbbl(float psw);
float tsturbbl(float pts);
float blcoeff(float blp, float uinf);

main()
{   fp=fopen("all31.dat","w+");
    fx=fopen("outvsx.dat","w+");
    fy=fopen("outvsy.dat","w+");
    fi=fopen("mr20modj.dat","r");   /* Wall Temp data for 21 Mar 97 PIV data set  */

/*  Enter the run conditions. */

    printf("\nEnter Redh: ");
    scanf("%f",&Redh);
    printf("\nEnter Rot: ");
    scanf("%f",&Rot);
    if(Rot==0.0) Rot=1e-10;
    printf("\nEnter Pin (Pa): ");
    scanf("%f",&Pin);
    printf("\nEnter Tin (K): ");
    scanf("%f",&Tin);
/*    printf("\nEnter Tu level for bl growth adjustment (e.g. 0.1):  ");
    scanf("%f",&Tuperc);  */
    Tuperc=0.1;

/*  The test section geometry in meters:  dh=hydraulic diameter
    and Rin is the inlet radius    */

    dh=0.01;
    Rin=0.401;

/*  J is the # of delta y intervals for discretizing the full channel */

    J=1500;
    dely=dh/J;

/*  Set the inlet gas conditions   */

    Rgas=287;          /*  Gas constant for air                  */
    specheat=1005.0; /*  Specific Heat in J/kgK at ref = 300K */

/*  Compute viscosity in kg/ms as a function of Tin  */

    visc=6.0145e-6 + 4.1402e-8*Tin;

/*  Compute inlet gas density assuming ideal gas behavior.  */

    densin=Pin/Tin/Rgas;
    kvisc=visc/densin;

/*  Set the channel inlet conditions (step=0) assuming plug flow */

    uin=Redh*kvisc/dh;
    uinftbl[0]=uin;

    wrad=Rot*uin/dh;
    wrps=wrad/(2.0*3.1415);

    for(j=0;j<=J;j++){
        usid[0][j]=uin;
        uls[0][j]=uin;
        uts[0][j]=uin;
        wts[0][j]=0.0;
        wls[0][j]=0.0;
        Tsid[0][j]=Tin;
        Tls[0][j]=Tin;
        Tts[0][j]=Tin;
        denssid[0][j]=densin;
        densls[0][j]=densin;
        densts[0][j]=densin;
        dwlsbuoy[j]=0.0;
        dwtsbuoy[j]=0.0;
    }
    Tavgls[0]=Tin;
    Tavgsid[0]=Tin;
    Tavgts[0]=Tin;
    uavgls[0]=uin;
    uavgsid[0]=uin;
    uavgts[0]=uin;
    davgls[0]=densin;
    davgsid[0]=densin;
    davgts[0]=densin;

    for(j=0;j<=J/2;j++){
        vsid[0][j]=0.0;
        vsidorig[0][j]=0.0;
        ucor[0][j]=0.0;
    }
    vorteye[0]=0.0;
    mbunmix=0.0;
    ysep=0.0;

/*  The program steps in the streamwise direction a distance
    equal to delx which is some fraction of the test section dh      */

/*    printf("\nEnter streamwise step distance in fraction of dh (0.1): ");
    scanf("%f",&c4);  */
    c4=0.1;
    delx=c4*dh;

/*    printf("\nEnter ucor duffusion factor (0.8): ");
    scanf("%f",&c6);  */
    c6=1.0;

/*  Initialize x=0.0 at inlet & other parameters.  The xequiv variables
    are used to track the "effective" x of each boundary layer which
    corresponds to its coriolis/buoyancy modified displacement thickness.  */

    dt[0]=delx/uin;
    x[0]=0.0;
    xequiv_ls[0]=x[0];
    xequiv_ts[0]=x[0];
    xequiv_sid[0]=x[0];
    ublthk_lsmax=0.0;
    ublthk_sidmax=0.0;
    ublthk_tsmax=0.0;

    printf("\nEnter # of steps down the passage, N:  ");
    scanf("%d",&N);

/*  Enter thermal data if desired.     */

    printf("\nConsider Buoyancy? 1=yes 0=no:  ");
    scanf("%d",&ansB);
    if(ansB==1){
        printf("\nSelect:  variable Tw & known qw (2), const qw (1), or const Tw (0) ");
        printf("\nfor the wall thermal boundary condition:  ");
        scanf("%d",&thermalbc);
        if(thermalbc==0){
            printf("\nEnter density ratio, dr=(Tw-Tin)/Tin :  ");
            scanf("%f",&densrat);
            printf("\nWall Temp = %.1f degrees above freestream",Tin*densrat);
            printf("\nEnter amplitude of Twall variation ls>sid>ts : ");
            scanf("%f",&amplit);
            for(n=0;n<=N;n++){
                Twsid[n]=Tin*densrat+Tin;
                Twls[n]=Tin*densrat+Tin+amplit;
                Twts[n]=Tin*densrat+Tin-amplit;
            }
        }
        else if(thermalbc==2){
            printf("\nReading in wall temperature vectors...");
            for(n=0;n<=100;n++) fscanf(fi,"%f %f %f %f %f",
                                      &int_0,&Twls[n],&int_1,&Twts[n],&Twsid[n]);
/*          for(n=0;n<=100;n++) printf("\n%.3f %.3f %.3f",Twls[n],Twsid[n],Twts[n]);  */
            int_0=0.0;
            int_1=0.0;
            int_2=0.0;
            for(n=0;n<=N;n++){
                int_0=int_0+Twls[n];
                int_1=int_1+Twsid[n];
                int_2=int_2+Twts[n];
            }
            printf("\nSide\tAvg Twall\tSide d.r.");
            printf("\nLS\t%.1f\t\t%.3f",int_0/(N+1),(int_0/(N+1)-Tin)/Tin);
            printf("\nSW\t%.1f\t\t%.3f",int_1/(N+1),(int_1/(N+1)-Tin)/Tin);
            printf("\nLTS\t%.1f\t\t%.3f",int_2/(N+1),(int_2/(N+1)-Tin)/Tin);
            densrat=((int_0+2.0*int_1+int_2)/4.0/(N+1)-Tin)/Tin;
            printf("\nNet d.r. = %.3f",densrat);
            printf("\nEnter percent of LS buoyant flow to unmix (0-100): ");
            scanf("%f",&percbls);

            qwls=9255.0;
            qwsid=10419.0;  /* average of sides 1 & 3 for J data  */
/*          qwsid=11355.0;  side 1 for J data  */
/*          qwsid=9482.0;  side 3 for J data  */
            qwts=10769.0;
/*          qwls=10000;
            qwsid=10000;
            qwts=10000;  */
        }
        else{
            printf("\nEnter wall heat flux [W/m2] : ");
            scanf("%f",&heatflux);
            for(n=0;n<=N;n++){
                Twls[n]=Tin;
                Twsid[n]=Tin;
                Twts[n]=Tin;
            }
        }
    }
    else{
        thermalbc=99;
        densrat=0.0;
        for(n=0;n<=N;n++){
            Twls[n]=Tin;
            Twsid[n]=Tin;
            Twts[n]=Tin;
        }
    }

/*  **********************************************************************  */
/*  ********nnnnnnnnn   START Streamwise 'n' Loop   nnnnnnnnnn***********  */

/*  Now start the 'n' loop that steps delx down the channel, computing
    the cross-flows and the changes in the streamwise profile with x.  */

/*  Set the initial values at n=1   */

    uinftbl[1]=uin;
    x[1]=x[0]+delx;
    xequiv_ls[1]=xequiv_ls[0]+delx;
    xequiv_sid[1]=xequiv_sid[0]+delx;
    xequiv_ts[1]=xequiv_ts[0]+delx;
    dt[1]=delx/uinftbl[1];

/*  Set the initial values for the profile power exponents    */

    pls=9.0;
    psw=9.0;
    pts=9.0;

    for(n=1;n<=N;n++){
        printf("\nn= %d ",n);

/*  TTTTTTTTTTTTTTT  BOUNDARY LAYER RELATIONS  TTTTTTTTTTTTTTTT  */

/*  Calculate the boundary layer momentum thickness using a power law profile.
    To account for acceleration, the integral is numerically calculated by summing.
    Then compute the displacement and boundary layer thicknesses.  */

        coeff=blcoeff(pls,uinftbl[n]);
        uinfpwr=((pls+3.0)*(3.0*pls+2)-2.0*pls)/(pls*(pls+1.0));
        int_0=0.0;
        for(j=1;j<=n;j++) int_0=int_0+pow(0.5*(uinftbl[j]+uinftbl[j-1]),uinfpwr)*
                                (xequiv_ls[j]-xequiv_ls[j-1]);
        umomthk_ls[n]=coeff*pow(int_0,(pls+1.0)/(pls+3.0));
        ublthk_ls[n]=umomthk_ls[n]/(pls/(pls+1.0)-pls/(pls+2.0));
        udispthk_ls[n]=ublthk_ls[n]*(1.0-pls/(pls+1.0));

/*  Likewise for the trailing side...  */

        coeff=blcoeff(pts,uinftbl[n]);
        uinfpwr=((pts+3.0)*(3.0*pts+2)-2.0*pts)/(pts*(pts+1.0));
        int_0=0.0;
        for(j=1;j<=n;j++) int_0=int_0+pow(0.5*(uinftbl[j]+uinftbl[j-1]),uinfpwr)*
                                (xequiv_ts[j]-xequiv_ts[j-1]);
        umomthk_ts[n]=coeff*pow(int_0,(pts+1.0)/(pts+3.0));
        ublthk_ts[n]=umomthk_ts[n]/(pts/(pts+1.0)-pts/(pts+2.0));
        udispthk_ts[n]=ublthk_ts[n]*(1.0-pts/(pts+1.0));
```

```
/*  And for the side wall...  */

        coeff=blcoeff(psw,uinftbl[n]);
        uinfpwr=((psw+3.0)*(3.0*psw+2)-2.0*psw)/(psw*(psw+1.0));
        int_0=0.0;
        for(j=1;j<=n;j++) int_0=int_0+pow(0.5*(uinftbl[j]+uinftbl[j-1]),uinfpwr)*
                                 (xequiv_sid[j]-xequiv_sid[j-1]);
        umomthk_sid[n]=coeff*pow(int_0,(psw+1.0)/(psw+3.0));
        ublthk_sid[n]=umomthk_sid[n]/(psw/(psw+1.0)-psw/(psw+2.0));
        udispthk_sid[n]=ublthk_sid[n]*(1.0-psw/(psw+1.0));

/*  With the boundary layer thickness estimates, I can use the result of the
    momentum integral equation to determine the actual profile shape for u and T.  */

        printf("\tdlQlsin dlQcore dlQtsout\tTavgls\tTavgsid\tTavgts\n");

/*  For thermalbc=2, I need an estimate of Q convected by coriolis vortex.
    To compute the integrals, use Tsid, vsid, and vorteye from last step.   */

        int_0=0.0;
        for(j=1;j<=(vorteye[n-1]/dely);j++) int_0=int_0+dely*
             0.5*(Tsid[n-1][j]+Tsid[n-1][j-1])*0.5*fabs(vsid[n-1][j]+vsid[n-1][j-1]);
        if(n==1) denssidmni=densin;
        else{
            denssidmni=0.0;
            for(j=1;j<=(vorteye[n-1]/dely);j++) denssidmni=denssidmni
                    +0.5*(denssid[n-1][j]+denssid[n-1][j])*dely;
            denssidmni=denssidmni/(j*dely);
        }
        delQlsin[n]=specheat*densin*delx*int_0;

        int_0=0.0;
        for(j=(vorteye[n-1]/dely);j<=(0.5*dh/dely);j++) int_0=int_0+
            0.5*fabs(vsid[n-1][j]+vsid[n-1][j-1])*dely;
        delQcore[n]=specheat*densin*delx*Tin*int_0;

        lsturbbl(pls);

/*  Likewise for the trailing side...  */

        int_0=0.0;
        for(j=1;j<=(ublthk_ts[n-1]/dely);j++) int_0=int_0+dely*
             0.5*(Tts[n-1][j]+Tts[n-1][j-1])*0.5*fabs(wts[n-1][j]+wts[n-1][j-1]);
        delQtsout[n]=specheat*densin*delx*int_0;
        printf("\t%.3g\t%.3g\t%.3g",delQlsin[n],delQcore[n],delQtsout[n]);

        tsturbbl(pts);

/*  DDDDDDDDDDDDDDD  DIFFUSE STREAMWISE CORIOLIS     DDDDDDDDDDDDDDDDDDDDDDD  */

/*  With the usw profile for this step, add the streamwise coriolis flow
    left over from last step.  If n=1 there is nothing to carry forward.
    Before adding the two profiles the coriolis flow is diffused away from
    the wall over the timestep.  */

        for(j=0;j<=J/2;j++) ucor[n][j]=0.0;

        if(n>1){
            jdiff=c6*Tuperc*uinftbl[n]*dt[n]/dely;
            for(j=0;j<=J/2;j++){
                if(ucor[n-1][j]>0.0){
                    if((j+jdiff)>J/2) gmin=J/2-jdiff;
                    else gmin=j;
                    for(g=gmin;g<=(gmin+jdiff);g++)
                                    ucor[n][g]=ucor[n][g]+ucor[n-1][j]/(jdiff+1.0);
                }
            }
        }

        int_0=0.0;
        for(j=0;j<=J/2;j++) int_0=int_0+ucor[n-1][j];
/*      printf("\tint0=%g",int_0);  */

/*  Finally compute the profile for the side wall...then add ucor to it.  */

        sidturbbl(psw);

        for(j=0;j<=J/2;j++) usid[n][j]=usid[n][j]+ucor[n][j];

        printf("\t%.2f\t%.2f\t%.2f  ",Tavgls[n],Tavgsid[n],Tavgts[n]);
        printf("\tnTls=%.3f nTsid=%.3f nTts=%.3f",nTls,nTsid,nTts);

/*  If any of the b.l.s are shrinking, I want to be sure and integrate
    out to the maximum bl so as not to lose any prior information.  */

        if(ublthk_ls[n]>ublthk_lsmax) ublthk_lsmax=ublthk_ls[n];
        if(ublthk_sid[n]>ublthk_sidmax) ublthk_sidmax=ublthk_sid[n];
        if(ublthk_ts[n]>ublthk_tsmax) ublthk_tsmax=ublthk_ts[n];

/*  CCCCCCCCCCC    BEGIN V CORIOLIS COMPONENT COMPUTATION   CCCCCCCCCCCCCCC  */

/*  Compute the mid-step density for later use  */

        for(j=0;j<=J;j++){
            denssidm[j]=0.5*(denssid[n][j]+denssid[n-1][j]);
            denslsm[j]=0.5*(densls[n][j]+densls[n-1][j]);
            denstsm[j]=0.5*(densts[n][j]+densts[n-1][j]);
        }

/*  Now with the streamwise velocity profiles, I can compute the additional
    coriolis-induced transverse velocity over this streamwise step.  Since
    usid is available only at the "n" nodes, use an average of the n and
    n-1 profiles.  Since I assume passage symmetry, only perform Coriolis
    adjustment for J/2 steps. This produces vsid<0.  */

        for(j=0;j<=J/2;j++) vsid[n][j]=vsidorig[n-1][j]+2.0*wrad
                            *(0.5*(usid[n][j]+usid[n-1][j])*denssidm[j]
                            /(0.5*(uinftbl[n]+uinftbl[n-1])*densin)-1)*delx;

/*  This induced transverse velocity profile must satisfy continuity.
    Compute the line integral of denssid*vsid and find the mean value.
    Since vsid is actually between "n" and "n-1", use a mid-step density
    profile as well.   */

        int_0=0.0;
        for(j=1;j<=J/2;j++) int_0=int_0+0.5*(vsid[n][j]+vsid[n][j-1])
                                *0.5*(denssidm[j]+denssidm[j-1])*dely;
        int_0=int_0/(0.5*dh);

/*  int_0 is the mean line massflow quantity.  Subtract it from vsid to
    create a profile which now satisfies continuity.  Retain this profile
    as vsidorig to avoid kinks in the profile at subsequent steps.  */

        for(j=0;j<=J/2;j++){
            vsid[n][j]=(vsid[n][j]*denssidm[j]-int_0)/denssidm[j];
            vsidorig[n][j]=vsid[n][j];
        }

/*  The next step is to make the velocity at the wall go to zero.
    Before reshaping the near wall profile, find the crossover point
    (or vortex "eye") and compute the massflow for y<yeye.   */

        mdotsw=0.0;
        for(j=1;j<=J/2;j++){
            if(vsid[n][j]>=0.0) break;
            mdotsw=mdotsw+0.5*fabs(vsid[n][j]+vsid[n][j-1])
                      *0.5*(denssidm[j]+denssidm[j-1])*dely;
        }
        jeye=j;
        vorteye[n]=jeye*dely;

/*  Now determine the shear (slope) at yeye using center difference. */

        j=jeye;
        dvsiddyeye=(vsid[n][j+1] - vsid[n][j-1])/(2*dely);

/*  To proceed in reshaping the vsid profile, I'll need a mean density
    for both regions  0<y<yeye  and yeye<y<dh/2 .     */

        denssidmni=0.0;
        for(j=1;j<=jeye;j++) denssidmni=denssidmni+denssidm[j];
        denssidmni=denssidmni/jeye;

        denssidmno=0.0;
        for(j=jeye+1;j<=J/2;j++) denssidmno=denssidmno+denssidm[j];
        denssidmno=denssidmno/(J/2-jeye);

/*  Reshape the profile for y<yeye into a form
          vsid = constant * (y/yeye)^m * (1-(y/yeye)^n)
    The constant is set by matching the slope at the vortex eye.  This
    gives constant = -(dv/dy@y=yeye) * yeye / n .
    The power 'm' is set to 0.5 which gives a full profile.
    Finally, 'n' is set by preserving the mass flow in the crossflow.  */

/*  Now determine the exponent "n" which will conserve mdotsw  */

        npow=(2.0/3.0)*denssidmni*pow(vorteye[n],2.0)*dvsiddyeye/mdotsw-1.5;

/*  If npow<0 this profile explodes at y=0.  So in this case, set npow to
    a minimum value of 0.01 and compute the slope at yeye that corresponds
    to this npow AND conserves massflow.  */

        if(npow<=0.0){
            npow=1.0;
            dvsiddyeye=1.5*(npow+1.5)*mdotsw/denssidmni/pow(vorteye[n],2.0);
        }
        vsidmax[n]=0.0;
        for(j=0;j<=jeye;j++){
            vsid[n][j]=-vorteye[n]*dvsiddyeye*pow((j/jeye),0.5)*
                            (1-pow((j/jeye),npow))/npow;
            if(vsid[n][j]<vsidmax[n])  vsidmax[n]=vsid[n][j];
        }

/*  For the region yeye<y<dh/2 reshape vsid into a profile or the form:
          vsid = constant * (y-dh/2/yeye-dh/2)^m * (1-(y-dh/2/yeye-dh/2))
    The constant is set by matching the slope at the vortex eye.  This
    gives constant = (dh/2-yeye) * (dv/dy@y=yeye) .
    The power 'm' is set by preserving the mass flow in the crossflow.  */

        mpow=-1.5+pow(0.25+denssidmno*dvsiddyeye*pow((dh/2.0-vorteye[n]),2.0)/
                        mdotsw,0.5);
        for(j=jeye;j<=J/2;j++) vsid[n][j]=(dh/2.0-vorteye[n])*dvsiddyeye*
                               pow((j-J/2.0)/(jeye-J/2.0),mpow)*
                                (1.0-(j-J/2.0)/(jeye-J/2.0));

/*  CCCCCCCCCCCCCC    END V CORIOLIS COMPONENT COMPUTATION        CCCCCCCCCCCCCCC  */

/*  CCCCCCCCCCCCCC    BEGIN  U CORIOLIS COMPONENT COMPUTATION     CCCCCCCCCCCCCCC  */

/*  There is a component of the Coriolis acceleration which acts on the
    streamwise momentum equation.  The contribution of this acceleration
    for this time step is computed.  */

/*      for(j=0;j<=J/2;j++) ducor[j]=0.0; */
        for(j=0;j<=J/2;j++) ducor[j]=-2.0*wrad*delx*vsid[n][j]*denssidm[j]
                            /(0.5*(uinftbl[n]+uinftbl[n-1])*densin);

/*  In the region where ducor>0, diffuse it for 1/2 a timestep.   */

        for(j=0;j<=J/2;j++) ducorp[j]=0.0;
        jdiff=c6*Tuperc*uinftbl[n]*dt[n]*0.5/dely;
        for(j=0;j<=J/2;j++){
            if(ducor[j]>0.0){
                if((j+jdiff)>J/2) gmin=J/2-jdiff;
                else gmin=j;
                for(g=gmin;g<=(gmin+jdiff);g++)
                        ducorp[g]=ducorp[g]+ducor[j]/(jdiff+1.0);
            }
        }

/*  Now add this step's ducorp to the diffused ucor from last step.  */

        for(j=0;j<=J/2;j++) ucor[n][j]=ucor[n][j]+ducorp[j];

/*  In the region where ducor<0, this core deceleration will push
    core fluid toward the walls, thus thinning the boundary layers.  Model
    this as a displacement thickness reduction for the boundary layers --
    which will in turn reduce uinf. (sumcorshft>0) */

        sumcorshft=0.0;
        for(j=1;j<=J/2;j++){
            if(ducor[j]<0){
                sumcorshft=sumcorshft+dely*0.5*fabs(ducor[j]+ducor[j-1])
                           *0.5*(denssidm[j]+denssidm[j-1])
                           /(0.5*(uinftbl[n]+uinftbl[n-1])*densin);
            }
        }

/*  CCCCCCCCCCCCCC    END U CORIOLIS COMPONENT COMPUTATION      CCCCCCCCCCCCCCC */

/*  BBBBBBBBBBBBBB    BEGIN U BUOYANCY COMPONENT COMPUTATION      BBBBBBBBBBBBBBB */
```

```
        if(ansB==1){

/*  Compute the additional buoyancy flow for this step.  */

            for(j=0;j<=J;j++){
                dubuoyls[j]=(Rin+x[n]-delx/2)*wrad*wrad*(denslsm[j]/densin-1.0)
                            *delx/(0.5*(uinftbl[n]+uinftbl[n-1]));
                dubuoysid[j]=(Rin+x[n]-delx/2)*wrad*wrad*(denssidm[j]/densin-1.0)
                            *delx/(0.5*(uinftbl[n]+uinftbl[n-1]));
                dubuoyts[j]=(Rin+x[n]-delx/2)*wrad*wrad*(denstsm[j]/densin-1.0)
                            *delx/(0.5*(uinftbl[n]+uinftbl[n-1]));
            }

/*  Find maximum boundary layer end nodes before performing integration. */

            jblls=ublthk_lsmax/dely;
            jblsid=ublthk_sidmax/dely;
            jblts=ublthk_tsmax/dely;

/*  This additional buoyant flow must satisfy continuity for this time step.
    Use a trapezoidal rule to perform the numerical integration and find
    the average of the dubuoy*dens flow over the cross-section.      */

            int_0=0.0;
            for(j=1;j<=jblls;j++) int_0=int_0+(0.5*dh-ublthk_sidmax*j/jblls)
                           *0.5*(dubuoyls[j]+dubuoyls[j-1])
                           *0.5*(denslsm[j]+denslsm[j-1])*dely;
            for(j=1;j<=jblsid;j++) int_0=int_0+(dh-ublthk_lsmax*j/jblsid-
                           ublthk_tsmax*j/jblsid)
                           *0.5*(dubuoysid[j]+dubuoysid[j-1])
                           *0.5*(denssidm[j]+denssidm[j-1])*dely;
            for(j=1;j<=jblts;j++) int_0=int_0+(0.5*dh-ublthk_sidmax*j/jblts)
                           *0.5*(dubuoyts[j]+dubuoyts[j-1])
                           *0.5*(denstsm[j]+denstsm[j-1])*dely;

            int_0=int_0/(0.5*dh*dh);

/*  Subtract this offset from the buoyancy velocity profiles. */

            for(j=0;j<=J;j++){
                dubuoyls[j]=(dubuoyls[j]*denslsm[j]-int_0)/denslsm[j];
                dubuoysid[j]=(dubuoysid[j]*denssidm[j]-int_0)/denssidm[j];
                dubuoyts[j]=(dubuoyts[j]*denstsm[j]-int_0)/denstsm[j];
            }

/*  Compute the momentum deficit of each dubuoy<0 region to be added later
    to the displacement thickness.   */

            sumdefls_bmix=0.0;
            for(j=1;j<=J;j++){
                if(dubuoyls[j]<0){
                    sumdefls_bmix=sumdefls_bmix+dely*0.5*fabs(dubuoyls[j]+dubuoyls[j-1])
                           *0.5*(denslsm[j]+denslsm[j-1])
                           /(0.5*(uinftbl[n]+uinftbl[n-1])*densin);
                }
            }

/*  Retain some of this buoyant flow on the leading wall to create a separation
    bubble if desired.  */

            mbunmix=mbunmix+(percbls/100.0)*sumdefls_bmix
                    *0.5*(uinftbl[n]+uinftbl[n-1])*densin;
            sumdefls_bmix=(1.0-percbls/100.0)*sumdefls_bmix;

            sumdefsid_bmix=0.0;
            for(j=1;j<=J/2;j++){
                if(dubuoysid[j]<0){
                    sumdefsid_bmix=sumdefsid_bmix+dely*0.5*fabs(dubuoysid[j]+dubuoysid[j-1])
                           *0.5*(denssidm[j]+denssidm[j-1])
                           /(0.5*(uinftbl[n]+uinftbl[n-1])*densin);
                }
            }

            sumdefts_bmix=0.0;
            for(j=1;j<=J;j++){
                if(dubuoyts[j]<0){
                    sumdefts_bmix=sumdefts_bmix+dely*0.5*fabs(dubuoyts[j]+dubuoyts[j-1])
                           *0.5*(denstsm[j]+denstsm[j-1])
                           /(0.5*(uinftbl[n]+uinftbl[n-1])*densin);
                }
            }
/*  In the region where dubuoy>0, this core acceleration will pull
    core fluid away from the walls, thinning the boundary layers.  Model
    this as a displacement thickness addition for the boundary layers --
    which will in turn increase uinf.  Since the flow satisfies continuity
    just double the corresponding sumdef_bmix.  */

            sumdefls_bmix=2.0*sumdefts_bmix;
            sumdefsid_bmix=2.0*sumdefsid_bmix;
            sumdefts_bmix=2.0*sumdefts_bmix;
        }
        else{
            sumdefls_bmix=0.0;
            sumdefsid_bmix=0.0;
            sumdefts_bmix=0.0;
        }

/*  BBBBBBBBBBBBBB    END U BUOYANCY COMPONENT COMPUTATION     BBBBBBBBBBBBBB */

/*  BBBBBBBBBBBBBB    BEGIN W BUOYANCY COMPONENT COMPUTATION     BBBBBBBBBBBBBB */

        if(ansB==1){

/*  Compute the secondary buoyancy flow for this step, and add it to the
    cumulative velocity for both the leading and trailing boundary layers.  */

            jblls=ublthk_ls[n]/dely;
            dwlsmn=0.0;
            for(j=0;j<=jblls;j++){
                dwlsbuoy[j]=(0.5*dh-dely*j)*wrad*wrad*(1.0-denslsm[j]/densin)
                           *delx/(0.5*(uinftbl[n]+uinftbl[n-1]))+dwlsbuoy[j];
                dwlsmn=dwlsmn+dwlsbuoy[j]*dely;
              }
            dwlsmn=dwlsmn/((jblls+1.0)*dely);

            jblts=ublthk_ts[n]/dely;
            dwtsmn=0.0;
            for(j=0;j<=jblts;j++){
                dwtsbuoy[j]=(0.5*dh-dely*j)*wrad*wrad*(1.0-denstsm[j]/densin)
                           *delx/(0.5*(uinftbl[n]+uinftbl[n-1]))+dwtsbuoy[j];
                dwtsmn=dwtsmn+dwtsbuoy[j]*dely;
              }
            dwtsmn=dwtsmn/((jblts+1.0)*dely);

/*  These secondary buoyancy flows act as wall injection and tend to increase the
    boundary layer displacement thickness.  */

            denslsmni=0.0;
            for(j=1;j<=jblls;j++) denslsmni=denslsmni+denslsm[j];
            denslsmni=denslsmni/jblls;

            deflswbuoy=(denslsmni*dwlsmn/(densin*0.5*(uinftbl[n]+uinftbl[n-1])))*
                       delx*(1.0-pls/(pls+1.0))/(pls/(pls+1.0)-pls/(pls+2.0));

            denstsmni=0.0;
            for(j=1;j<=jblts;j++) denstsmni=denstsmni+denstsm[j];
            denstsmni=denstsmni/jblts;

            deftswbuoy=(denstsmni*dwtsmn/(densin*0.5*(uinftbl[n]+uinftbl[n-1])))*
                       delx*(1.0-pts/(pts+1.0))/(pts/(pts+1.0)-pts/(pts+2.0));
        }
        else{
            deflswbuoy=0.0;
            deftswbuoy=0.0;
        }

/*  BBBBBBBBBBBBBB    END W BUOYANCY COMPONENT COMPUTATION     BBBBBBBBBBBBBB */

/*  LLLLLLLLLLLLLL    ASSESS AFFECT ON LEADING WALL          LLLLLLLLLLLLLL */

/*  Now that the sidewall secondary mass flow has been computed, compute the
    leading side secondary velocity profile using continuity at the
    corner.  Assume a velocity profile:
        wls = constant * (y/blthick)^0.5 * (1 - y/blthick)^3
    The constant is determined by conserving mass.               */

        jblls=ublthk_ls[n]/dely;
        int_0=0.0;
        for(j=1;j<=jblls;j++)  int_0=int_0+0.5*(denslsm[j]+denslsm[j-1])*
                           pow((j-0.5)/jblls,0.5)*pow((1.0-(j-0.5)/jblls),3.0)*dely;
        wlsmax[n]=0.0;
        for(j=0;j<=jblls;j++){
            wls[n][j]=mdotsw*pow(j/jblls,0.5)*pow((1.0-j/jblls),3.0)/int_0;
            if(wls[n][j]>wlsmax[n])  wlsmax[n]=wls[n][j];
        }

/*  Now the 3 streamwise, u__(y), and 2 transverse, vsid(y) & wls(y),
    velocity profiles are in hand for this streamwise position.
    The next step is to see how much momentum deficit is convected to the
    leading side due to the coriolis-induced vortex flow during this last
    time step.  To do this, compute the net streamwise usid(y) momentum deficit
    located in the vsid(y)<0 part of the transverse velocity boundary
    layer.  This is the deficit that is convected to the leading side during
    this time step.  Again, since this is over the time step, dt, use
    average values of usid and vsid for this step.  (sumdefls>0)      */

        sumdefls=0.0;
        for(j=1;j<=jeye;j++){
            ratio=0.25*(usid[n][j]+usid[n-1][j]+usid[n][j-1]+usid[n-1][j-1])
                  *0.5*(denssidm[j]+denssidm[j-1])
                  /(0.5*(uinftbl[n]+uinftbl[n-1])*densin);
/*          if(ratio>1.0) ratio=1.0;  */
            sumdefls=sumdefls+(1.0-ratio)*dely*0.5*fabs(vsid[n][j]+vsid[n][j-1])*dt[n];
        }

/*  The momentum deficits from both coriolis and buoyancy flows add
    to the developing turbulent boundary layer displacement
    thickness to produce an amplified boundary layer on the leading side.
    Since the deficit from coriolis affects the entire leading side,
    sumdefls is divided by (dh/2-blthksid/2) .  The added deficit will also
    tend to thicken the b.l., decreasing the power exponent.  */

        coeff=Tuperc*ublthk_ls[n]/delx;
        pls=(udispthk_ls[n]/(1.0-pls/(pls+1.0)))/(udispthk_ls[n]+
            coeff*(sumdefls_bmix-0.25*sumcorshft+deflswbuoy
            +sumdefls/(0.5*dh-0.5*ublthk_sid[n])))-1.0;
        printf("\tpls= %.2f",pls);

        ublthk_ls[n]=(udisp thk_ls[n]+sumdefls_bmix-0.25*sumcorshft+deflswbuoy
                      +sumdefls/(0.5*dh-0.5*ublthk_sid[n]))/(1.0-pls/(pls+1.0));
        umomthk_ls[n]=ublthk_ls[n]*(pls/(pls+1.0)-pls/(pls+2.0));

/*  TTTTTTTTTTTTTT    ASSESS AFFECT ON TRAILING WALL          TTTTTTTTTTTTTTTTTT */

/*  The vortex flow also removes low momentum fluid from the trailing
    side wall.  This will decrease the trailing boundary layer thickness, since
    the exiting fluid is replaced with core fluid at uinf.  To account for
    this, employ the same method used with the leading wall's gain of
    momentum deficit.  Use continuity at the corner to compute a transverse
    velocity profile, since the mass flow must be conserved.  Here
    a laminar free convection profile is assumed from the trailing wall to the
    edge of the streamwise boundary layer thickness or yeye, whichever is
    larger.  */

        if(ublthk_ts[n]>vorteye[n]) jblts=ublthk_ts[n]/dely;
        else jblts=vorteye[n]/dely;

        int_0=0.0;
        for(j=1;j<=jblts;j++)  int_0=int_0+0.5*(denstsm[j]+denstsm[j-1])
                               *((j-0.5)/jblts)*pow((1.0-(j-0.5)/jblts),2.0)*dely;

        wtsmax[n]=0.0;
        for(j=0;j<=jblts;j++){
            wts[n][j]=-mdotsw*(j/jblts)*pow((1.0-j/jblts),2.0)/int_0;
            if(wts[n][j]<wtsmax[n]) wtsmax[n]=wts[n][j];
        }

/*  With wts computed, calculate the total momentum deficit lost
    from the trailing wall by this vortex flow.  (sumdefts>0)  */

        sumdefts=0.0;
        for(j=1;j<=jblts;j++){
            ratio=0.25*(uts[n][j]+uts[n-1][j]+uts[n][j-1]+uts[n-1][j-1])
                  *0 .5*(denstsm[j]+denstsm[j-1])
                  /(0.5*(uinftbl[n]+uinftbl[n-1])*densin);
/*          if(ratio>1.0) ratio=1.0;   */
            sumdefts=sumdefts+(1.0-ratio)*dely*0.5*fabs(wts[n][j]
                     +wts[n][j-1])*dt[n];
        }

/*  This momentum deficit subtracts from the developing turbulent
```

```
    boundary layer displacement thickness to produce a vortex decreased
    dislplacement thickness on the trailing side.  If needed, reduce
    this effect to prevent the boundary layer from disappearing.  */

        coeff=Tuperc*ublthk_ts[n]/delx;
        if((sumdefts/(0.5*dh-0.5*ublthk_sid[n])-sumdefts_bmix+0.25*sumcorshft
            -deftswbuoy)>udispthk_ts[n]*0.5){
            psw=(udispthk_ts[n]/(1.0-pts/(pts+1.0)))/(udispthk_ts[n]-
            coeff*udispthk_ts[n]*0.5)-1.0;
            udispthk_ts[n]=udispthk_ts[n]*0.5;
        }
        else{
            pts=(udispthk_ts[n]/(1.0-pts/(pts+1.0)))/(udispthk_ts[n]+
                coeff*(sumdefts_bmix+deftswbuoy-0.25*sumcorshft
                -sumdefts/(0.5*dh-0.5*ublthk_sid[n])))-1.0;
            udispthk_ts[n]=udispthk_ts[n]+sumdefts_bmix-0.25*sumcorshft+deftswbuoy
                            -(sumdefts/(0.5*dh-0.5*ublthk_sid[n]));
        }

        ublthk_ts[n]=udispthk_ts[n]/(1.0-pts/(pts+1.0));
        umomthk_ts[n]=ublthk_ts[n]*(pts/(pts+1.0)-pts/(pts+2.0));

/*  SSSSSSSSSSSSSSSS    ASSESS AFFECT ON SIDE WALL          SSSSSSSSSSSSSSSSSS */

/*  The side wall loses some momentum deficit as well (the difference
    between the trailing wall loss and the leading wall gain).       Subtract
    this from the displacement thickness there and compute the new profiles.  */

        coeff=Tuperc*ublthk_sid[n]/delx;
        psw=(udispthk_sid[n]/(1.0-psw/(psw+1.0)))/(udispthk_sid[n]+
            coeff*(sumdefsid_bmix-0.5*sumcorshft-(sumdefls-sumdefts)
            /(dh-ublthk_ls[n]/2.0-ublthk_ts[n]/2.0)))-1.0;

        printf("\tpsw= %.2f",psw);
        printf("\tpts= %.2f",pts);

        ublthk_sid[n]=(udispthk_sid[n]+sumdefsid_bmix-0.5*sumcorshft
                      -(sumdefls-sumdefts)/(dh-ublthk_ls[n]/2.0-ublthk_ts[n]/2.0))
                      /(1.0-psw/(psw+1.0));
        umomthk_sid[n]=ublthk_sid[n]*(psw/(psw+1.0)-psw/(psw+2.0));

/*  Update all of the profiles based on these changes in displacement thickness */

/*  Again for thermalbc=2, I need an estimate of Q convected by coriolis vortex.
    To compute the integrals, use the most recent Tsid, vsid, and vorteye.
    Still use the mean density from last step from Boussinesque approx.     */

        int_0=0.0;
        for(j=1;j<=(vorteye[n]/dely);j++) int_0=int_0+dely*
               0.25*(Tsid[n][j]+Tsid[n][j-1]+Tsid[n-1][j]+Tsid[n-1][j-1])
               *0.5*fabs(vsid[n][j]+vsid[n][j-1]);
        if(n==1) denssidmni=densin;
        else{
            denssidmni=0.0;
            for(j=1;j<=(vorteye[n-1]/dely);j++) denssidmni=denssidmni
                     +0.5*(denssid[n-1][j]+denssid[n-1][j])*dely;
            denssidmni=denssidmni/(j*dely);
        }
        delQlsin[n]=specheat*densin*delx*int_0;

        int_0=0.0;
        for(j=(vorteye[n]/dely);j<=(0.5*dh/dely);j++) int_0=int_0+
            0.5*fabs(vsid[n][j]+vsid[n][j-1])*dely;
        delQcore[n]=specheat*densin*delx*Tin*int_0;

        lsturbbl(pls);

        int_0=0.0;
        for(j=1;j<=(ublthk_ts[n]/dely);j++) int_0=int_0+dely*
               0.25*(Tts[n][j]+Tts[n][j-1]+Tts[n-1][j]+Tts[n-1][j-1])
               *0.5*fabs(wts[n][j]+wts[n][j-1]);
        delQtsout[n]=specheat*densin*delx*int_0;

        tsturbbl(pts);

        sidturbbl(psw);

        printf("\n\t%.3g\t%.3g\t%.3g",delQlsin[n],delQcore[n],delQtsout[n]);
        printf("\t%.2f\t%.2f\t%.2f",Tavgls[n],Tavgsid[n],Tavgts[n]);
/*  Now add the final ucor to usid for the uinf acceleration computation    */
```

```
        for(j=0;j<=J/2;j++) usid[n][j]=usid[n][j]+ucor[n][j];

/*  These flow adjustments have altered the boundary layers on all 3 sides.
    To account for this, calculate the equivalent x of the turbulent boundary
    that would produce this adjusted boundary layer momentum thickness.  */

        coeff=blcoeff(pls,uinftbl[n]);
        uinfpwr=((pls+3.0)*(3.0*pls+2)-2.0*pls)/(pls*(pls+1.0));
        int_0=0.0;
        for(j=1;j<=n-1;j++) int_0=int_0+pow(0.5*(uinftbl[j]+uinftbl[j-1]),uinfpwr)*
                                (xequiv_ls[j]-xequiv_ls[j-1]);

        xequiv_ls[n]=xequiv_ls[n-1]+
                     (pow((umomthk_ls[n]/coeff),(pls+3.0)/(pls+1.0))-int_0)
                    /pow(0.5*(uinftbl[n]+uinftbl[n-1]),uinfpwr);

        coeff=blcoeff(pts,uinftbl[n]);
        uinfpwr=((pts+3.0)*(3.0*pts+2)-2.0*pts)/(pts*(pts+1.0));
        int_0=0.0;
        for(j=1;j<=n-1;j++) int_0=int_0+pow(0.5*(uinftbl[j]+uinftbl[j-1]),uinfpwr)*
                                (xequiv_ts[j]-xequiv_ts[j-1]);

        xequiv_ts[n]=xequiv_ts[n-1]+
                     (pow((umomthk_ts[n]/coeff),(pts+3.0)/(pts+1.0))-int_0)
                    /pow(0.5*(uinftbl[n]+uinftbl[n-1]),uinfpwr);

        coeff=blcoeff(psw,uinftbl[n]);
        uinfpwr=((psw+3.0)*(3.0*psw+2)-2.0*psw)/(psw*(psw+1.0));
        int_0=0.0;
        for(j=1;j<=n-1;j++) int_0=int_0+pow(0.5*(uinftbl[j]+uinftbl[j-1]),uinfpwr)*
                                (xequiv_sid[j]-xequiv_sid[j-1]);

        xequiv_sid[n]=xequiv_sid[n-1]+
                     (pow((umomthk_sid[n]/coeff),(psw+3.0)/(psw+1.0))-int_0)
                    /pow(0.5*(uinftbl[n]+uinftbl[n-1]),uinfpwr);

/*  Before using continuity to obtain the core acceleration, determine the
    extent of the separation bubble created by the unmixed buoyancy flow
    on the leading side (if any was specified).  */

        int_0=0.0;
        for(j=1;j<=J;j++){
            int_0=int_0+0.5*(uls[n][j]+uls[n][j-1])*0.5*(densls[n][j]+densls[n][j-1])*dely;
            if(int_0>mbunmix) break;
        }

        if(j==1) ysep=0.0;
        else ysep=(j-0.5)*dely;

/*  The growing boundary layers on all 4 sides will cause
    a blockage in the passage and accelerate the core flow.
    Apply continuity on the 1/2 passage (assuming symmetry). Compute the
    integral numerically.  Since the boundary layers merge in the corners,
    use a "sharing" of deficit in the corners.  */

        jblls=(ublthk_ls[n]+ysep)/dely;
        jblsid=ublthk_sid[n]/dely;
        jblts=ublthk_ts[n]/dely;

        Aint_ls=0.0;
        jsep=ysep/dely;
        for(j=1;j<=jblls;j++){
            if(j<=jsep) Aint_ls=Aint_ls+(0.5*dh-ublthk_sid[n]*(j-0.5)/jblls)*dely;
            else Aint_ls=Aint_ls+(0.5*dh-ublthk_sid[n]*(j-0.5)/jblls)*dely
                        *(1.0-0.5*(densls[n][j-jsep]+densls[n][j-jsep-1])*
                        0.5*(uls[n][j-jsep]+uls[n][j-jsep-1])/(uinftbl[n]*densin));
        }

        Aint_ts=0.0;
        for(j=1;j<=jblts;j++) Aint_ts=Aint_ts
          +(0.5*dh-ublthk_sid[n]*(j-0.5)/jblts)*dely
            *(1.0-0.5*(densts[n][j]+densts[n][j-1])*
             0.5*(uts[n][j]+uts[n][j-1])/(uinftbl[n]*densin));

        Aint_sid=0.0;
        for(j=1;j<=jblsid;j++) Aint_sid=Aint_sid+
          (dh-(ublthk_ls[n]+ysep)*(j-0.5)/jblsid-ublthk_ts[n]*(j-0.5)/jblsid)
            *dely*(1.0-0.5*(denssid[n][j]+denssid[n][j-1])*
                      0.5*(usid[n][j]+usid[n][j-1])/(uinftbl[n]*densin));

        uinftbl[n]=uin*(dh*0.5)*dh/(dh*0.5*dh-Aint_ls-Aint_sid-Aint_ts);
```

```
/*  This uinftbl and the 3 xequiv are the only information passed to the
    next step.  To be consistent, use these to recompute the boundary
    layer profiles and statistics before printing.  */

        coeff=blcoeff(psw,uinftbl[n]);
        uinfpwr=((psw+3.0)*(3.0*psw+2)-2.0*psw)/(psw*(psw+1.0));
        int_0=0.0;
        for(j=1;j<=n;j++) int_0=int_0+pow(0.5*(uinftbl[j]+uinftbl[j-1]),uinfpwr)*
                                (xequiv_sid[j]-xequiv_sid[j-1]);
        umomthk_sid[n]=coeff*pow(int_0,(psw+1.0)/(psw+3.0));
        ublthk_sid[n]=umomthk_sid[n]/(psw/(psw+1.0)-psw/(psw+2.0));
        udispthk_sid[n]=ublthk_sid[n]*(1.0-psw/(psw+1.0));

        coeff=blcoeff(pls,uinftbl[n]);
        uinfpwr=((pls+3.0)*(3.0*pls+2)-2.0*pls)/(pls*(pls+1.0));
        int_0=0.0;
        for(j=1;j<=n;j++) int_0=int_0+pow(0.5*(uinftbl[j]+uinftbl[j-1]),uinfpwr)*
                                (xequiv_ls[j]-xequiv_ls[j-1]);
        umomthk_ls[n]=coeff*pow(int_0,(pls+1.0)/(pls+3.0));
        ublthk_ls[n]=umomthk_ls[n]/(pls/(pls+1.0)-pls/(pls+2.0));
        udispthk_ls[n]=ublthk_ls[n]*(1.0-pls/(pls+1.0));

        coeff=blcoeff(pts,uinftbl[n]);
        uinfpwr=((pts+3.0)*(3.0*pts+2)-2.0*pts)/(pts*(pts+1.0));
        int_0=0.0;
        for(j=1;j<=n;j++) int_0=int_0+pow(0.5*(uinftbl[j]+uinftbl[j-1]),uinfpwr)*
                                (xequiv_ts[j]-xequiv_ts[j-1]);
        umomthk_ts[n]=coeff*pow(int_0,(pts+1.0)/(pts+3.0));
        ublthk_ts[n]=umomthk_ts[n]/(pts/(pts+1.0)-pts/(pts+2.0));
        udispthk_ts[n]=ublthk_ts[n]*(1.0-pts/(pts+1.0));

        int_0=0.0;
        for(j=1;j<=(vorteye[n]/dely);j++) int_0=int_0+dely*
               0.25*(Tsid[n][j]+Tsid[n][j-1]+Tsid[n-1][j]+Tsid[n-1][j-1])
               *0.5*fabs(vsid[n][j]+vsid[n][j-1]);
        if(n==1) denssidmni=densin;
        else{
            denssidmni=0.0;
            for(j=1;j<=(vorteye[n-1]/dely);j++) denssidmni=denssidmni
                    +0.5*(denssid[n-1][j]+denssid[n-1][j])*dely;
            denssidmni=denssidmni/(j*dely);
        }
        delQlsin[n]=specheat*densin*delx*int_0;

        int_0=0.0;
        for(j=(vorteye[n]/dely);j<=(0.5*dh/dely);j++) int_0=int_0+
            0.5*fabs(vsid[n][j]+vsid[n][j-1])*dely;
        delQcore[n]=specheat*densin*delx*Tin*int_0;

        lsturbbl(pls);

/*  Adjust uls for the separation bubble.  */

        if(ysep>0.0){
            for(j=J;j>=0;j--){
                if(j<=jsep) uls[n][j]=0.0;
                else uls[n][j]=uls[n][j-jsep];
            }
        }

        int_0=0.0;
        for(j=1;j<=(ublthk_ts[n]/dely);j++) int_0=int_0+dely*
               0.25*(Tts[n][j]+Tts[n][j-1]+Tts[n-1][j]+Tts[n-1][j-1])
               *0.5*fabs(wts[n][j]+wts[n][j-1]);
        delQtsout[n]=specheat*densin*delx*int_0;

        tsturbbl(pts);

        sidturbbl(psw);

        printf("\n\t%.3g\t%.3g\t%.3g",delQlsin[n],delQcore[n],delQtsout[n]);
        printf("\t%.2f\t%.2f\t%.2f",Tavgls[n],Tavgsid[n],Tavgts[n]);

/*  Compute the boundary layer's average density for use in next step.  */

        davgls[n]=(Pin/Rgas)*(1.0+nTls)/(Twls[n]+Tin*nTls);
        davgsid[n]=(Pin/Rgas)*(1.0+nTsid)/(Twsid[n]+Tin*nTsid);
        davgts[n]=(Pin/Rgas)*(1.0+nTts)/(Twts[n]+Tin*nTts);

/*  To assess the energy content of the flow at each step, I need an average
```

```
    fluid temp. */

        Tavg=(Tavgls[n]*(0.5*dh-0.5*ublthk_sid[n])*ublthk_ls[n]
             +Tavgts[n]*(0.5*dh-0.5*ublthk_sid[n])*ublthk_ts[n]
             +Tavgsid[n]*(dh-0.5*ublthk_ls[n]-0.5*ublthk_ts[n])*ublthk_sid[n]
             +Tin*(0.5*dh-ublthk_sid[n])*(dh-ublthk_ls[n]-ublthk_ts[n]))
             /(0.5*dh*dh);

        printf("\tTavg=%.1f",Tavg);

/*  Now estimate the wall heat flux necessary to generate this Tavg.  */

        qest=(densin*uin*0.5*dh*dh)*specheat*(Tavg-Tin)/(2.0*dh*x[n]);
        printf("  qest=%.0f",qest);

/*  Now add the final ucor to usw for the output   */

        for(j=0;j<=J/2;j++) usid[n][j]=usid[n][j]+ucor[n][j];

/*  Compute the boundary layer's mean streamwise velocity for use in next step.  */

        uavgls[n]=(pls/(pls+1.0))*uinftbl[n];
        uavgts[n]=(pts/(pts+1.0))*uinftbl[n];

        uavgsid[n]=0.0;
        for(j=1;j<=(ublthk_sid[n]/dely);j++) uavgsid[n]=uavgsid[n]+
                                               dely*0.5*(usid[n][j]+usid[n][j-1]);
        uavgsid[n]=uavgsid[n]/(j*dely);

/*  Check for constant massflow.  */

        jblls=(ublthk_ls[n]+ysep)/dely;
        jblsid=ublthk_sid[n]/dely;
        jblts=ublthk_ts[n]/dely;

        mdot=0.0;
        for(j=1;j<=jblls;j++) mdot=mdot
          +(0.5*dh-ublthk_sid[n]*(j-0.5)/jblls)*dely
            *0.5*(uls[n][j]+uls[n][j-1])*0.5*(densls[n][j]+densls[n][j-1]);

        for(j=1;j<=jblts;j++) mdot=mdot
          +(0.5*dh-ublthk_sid[n]*(j-0.5)/jblts)*dely
            *0.5*(uts[n][j]+uts[n][j-1])*0.5*(densts[n][j]+densts[n][j-1]);

        for(j=1;j<=jblsid;j++) mdot=mdot+
          (dh-(ublthk_ls[n]+ysep)*(j-0.5)/jblsid-ublthk_ts[n]*(j-0.5)/jblsid)
            *dely*0.5*(usid[n][j]+usid[n][j-1])*0.5*(denssid[n][j]+denssid[n][j-1]);

        mdot=(mdot+uinftbl[n]*densin*(0.5*dh-ublthk_sid[n])
              *(dh-(ublthk_ls[n]+ysep)-ublthk_ts[n]));
        mdotrat=mdot/(densin*uin*0.5*dh*dh);
        printf("  mdotrat2=%.3f",mdotrat);

/*  Is reverse flow predicted this step?  */

        for(j=1;j<=J;j++){
            if(uls[n][j]<0.0||usid[n][j]<0.0||uts[n][j]<0.0){
                fprintf(fp,"\nReverse flow on step n= %d",n);
                printf("\nReverse flow on step n= %d",n);
                break;
            }
        }

        if((ublthk_ts[n]+ublthk_ls[n])>=dh||ublthk_sid[n]>=(0.5*dh)) break;
        if(ublthk_ts[n]<=0.0) break;

/*  Use the freestream velocity and xequiv from this step to pass along
    to the next "n+1" step.   */

        uinftbl[n+1]=uinftbl[n];
        xequiv_sid[n+1]=xequiv_sid[n]+delx;
        xequiv_ls[n+1]=xequiv_ls[n]+delx;
        xequiv_ts[n+1]=xequiv_ts[n]+delx;
        x[n+1]=x[n]+delx;
        dt[n+1]=delx/uinftbl[n];

    }
    if(n<=N) printf("\nThe boundary layers merged in %d steps.",n);
    else printf("\nMade it all the way to %d without merging. ",n-1);
    if(n>N) n=N;
    N=n;
```

```
/*  @@@@@@@@@@@nnnnnnnnnnnnnnnnnn   END   nnnnnnnnnnnnnnnn@@@@@@@@@@@@@@  */

/*  ********The program is done. ******       */

/*  Print out the run parameters to the 'fp' datafile.  */

    fprintf(fp,"\nRedh= %.0f\tRot= %.3f\tdh= %.3f\tPin= %.0f",Redh,Rot,dh,Pin);
    fprintf(fp,"\nuin= %.2f\tTin= %.1f\tdensin= %.3f\twrps= %.2f",
                uin,Tin,densin,wrps);
    fprintf(fp,"\n");
    if(ansB==1){
        if(thermalbc==1)  fprintf(fp,"\nThermalbc is constant heat
                                    flux = %.0f ",heatflux);
        else if(thermalbc==2){
            fprintf(fp,"\nVariable Twall from datafile.");
            fprintf(fp,"\nDensity ratio = %.3f",densrat);
            fprintf(fp,"\nPercent of LS buoyant mix out = %.2f ",percbls);
        }
        else{
            fprintf(fp,"\nThermalbc is constant wall temperature");
            fprintf(fp,"\nDensity ratio = %.3f",densrat);
            fprintf(fp,"\nWall temps are Tls= %.1f Tsid= %.1f and Tts= %.1f",
                        Twls[0],Twsid[0],Twts[0]);
        }
    }
    else fprintf(fp,"No buoyancy is considered for this run. ");
    fprintf(fp,"\nTurbulent mixing constant is Tuperc= %.2f ",Tuperc);
    fprintf(fp,"\n");
    fprintf(fp,"\n%d cross-stream intervals fit on each b.l.",J);
    fprintf(fp,"\n%d streamwise intervals with step = %.2f*dh",N,c4);

/*  Print out the data.*/

    printf("\nEnter interval for printing nstep data:  ");
    scanf("%d",&nstep);

    fprintf(fp,"\n");
    fprintf(fp,"\nAll velocities normalized by uin");
    fprintf(fp,"\nAll lengths normalized by dh");
    fprintf(fp,"\n");
    fprintf(fp,"\nVelocities");
    fprintf(fp,"\nn x uinftbl wlsmax vsidmax wtsmax");
    for(n=1;n<=N;n++){
        remaind=n%nstep;
        if(remaind==0) fprintf(fp,"\n%d %.2f %.4f %.4f %.4f %.4f %.4f",
                n,x[n]/dh,uinftbl[n]/uin,wlsmax[n]/uin,vsidmax[n]/uin,wtsmax[n]/uin);
    }

/*  Print the bl thicknesses.  */

    fprintf(fp,"\n");
    fprintf(fp,"\nThe boundary layer thicknesses and vortex eye position are");
    fprintf(fp,"\nn  x ublthk_ls ublthk_sid ublthk_ts vorteye");
    for(n=1;n<=N;n++){
        remaind=n%nstep;
        if(remaind==0) fprintf(fp,"\n%d %.2f %.4f %.4f %.4f %.4f",
            n,x[n]/dh,ublthk_ls[n]/dh,ublthk_sid[n]/dh,
            ublthk_ts[n]/dh,vorteye[n]/dh);
    }

/*  Print to non-headered file for use in Matlab plots  */

    for(n=1;n<=N;n++){
        remaind=n%nstep;
        if(remaind==0) fprintf(fx,"\n%.2f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f",
            x[n]/dh,uinftbl[n]/uin,ublthk_ls[n]/dh,ublthk_sid[n]/dh,
            ublthk_ts[n]/dh,vorteye[n]/dh,wlsmax[n]/uin,vsidmax[n]/uin,
            wtsmax[n]/uin);
    }


/*  If constant heat flux case, print out wall temperatures.  */

    if(thermalbc==1){
        fprintf(fp,"\n");
        fprintf(fp,"\nThe wall temperatures for const q case are");
        fprintf(fp,"\nn  x Twls Twsid Twts ");
        Twavg=0.0;
        for(n=1;n<=N;n++){
            Twavg=Twavg+Twls[n]+2*Twsid[n]+Twts[n];
            remaind=n%nstep;
            if(remaind==0) fprintf(fp,"\n%d %.2f %.4f %.4f %.4f",
                    n,x[n]/dh,Twls[n],Twsid[n],Twts[n]);
        }
        Twavg=Twavg/N/4.0;
        densrat=(Twavg-Tin)/Twavg;
        fprintf(fp,"\nConst q case:  densrat based on Twavg is: %f ",densrat);
    }

/*  Print b.l. data for plotting.  */

        do{
            printf("\nThere are N=%d time intervals before the model stopped",N);
            printf("\nEnter n <= N for b.l. data: ");
            scanf("%d",&n);
        }while(n>N||n<=0);

        fprintf(fp,"\n");
        fprintf(fp,"\nbl data at x/dh= %.2f",x[n]/dh);
        fprintf(fp,"\nzlsts ulsts wlsts ysid usid vsid");
        interv=10;  /*  The printing interval for boundary layer data  */

        for(j=0;j<=J;j++){
            remaind=j%interv;
            if(remaind==0) fprintf(fp,"\n%.4f %.6f %.6f %.4f %.6f %.6f",
                        (j*dely/dh-0.5),uls[n][j]/uin,wls[n][j]/uin,j*dely/dh,
                                    usid[n][j]/uin,vsid[n][j]/uin);
        }
        fprintf(fp,"\n");
        for(j=J;j>=0;j--){
            remaind=j%interv;
            if(remaind==0) fprintf(fp,"\n%.4f %.6f %.6f %.4f %.6f %.6f",
                        (0.5-j*dely/dh),uts[n][j]/uin,wts[n][j]/uin,(1.0-j*dely/dh),
                                    usid[n][j]/uin,vsid[n][j]/uin);
        }

/*  Print a non-headered profile file for Matlab   */

        for(j=0;j<=J;j++){
            remaind=j%interv;
            if(remaind==0) fprintf(fy,"\n%.4f %.6f %.6f %.6f %.6f %.6f %.6f %.6f",
                        j*dely/dh,uls[n][j]/uin,usid[n][j]/uin,uts[n][j]/uin,
                        wls[n][j]/uin,vsid[n][j]/uin,wts[n][j]/uin,ucor[n][j]/uin);
        }

/*  Close the 'fp' data file  */

    fprintf(fp,"\nThat's all folks");
    fclose(fp);
    fclose(fx);
    fclose(fy);
    fclose(fi);

    do{
        printf("\nPIV program data needed? 1=y, 0=n: ");
        scanf("%d",&ans);
    }while(ans>1);

/*  Print out the data for the PIV program  */

    if(ans==1){
        fq=fopen("jpbPVI10.dat","w+");
        printf("\nEnter beginning and ending N profiles needed (Beg End): ");
        scanf("%d %d",&Beg,&End);

        fprintf(fq,"\n%f %f %f %f %f",Redh,Rot,densrat,amplit,wrps);
        fprintf(fq,"\n%d %d %f %f",J,(End-Beg+1),dh,Rin);

        fprintf(fq,"\n");
        for(n=Beg;n<=End;n++){
            fprintf(fq,"\n%.6f %.6f %.6f %.6f",x[n]/dh,dely/dh,
                        dely/dh,dely/dh);
        }

        fprintf(fq,"\n");
        for(n=Beg;n<=End;n++){
            fprintf(fq,"\n%.6f %.6f %.6f %.6f",
                    ublthk_ls[n]/dh,ublthk_sid[n]/dh,ublthk_ts[n]/dh,
                    vorteye[n]/dh);
        }
        fprintf(fq,"\n");
        for(n=Beg;n<=End;n++){
            for(j=0;j<=J;j++){
                fprintf(fq,"\n%.6f %.6f %.6f %.6f %.6f %.6f",
                uls[n][j],usid[n][j],uts[n][j],wls[n][j],vsid[n][j],wts[n][j]);
            }
            fprintf(fq,"\n");
        }
        fprintf(fq,"\nThat's all folks");
        fclose(fq);
    }

    return(0);
}


/****************************************************************/
/****************************************************************/
/*  This subroutine computes the temperature profiles based on
    an application of energy conservation in the passage for the
    constant heat flux case.  */

energcons()
{   int j;
    float int_0,int_1;

/*  There are 2 integrals that need to be computed to apply energy
    conservation.  */

    jblls=ublthk_lsmax/dely;
    jblsid=ublthk_sidmax/dely;
    jblts=ublthk_tsmax/dely;

    int_1=0.0;
    for(j=1;j<=jblls;j++){
        if(((j-0.5)*dely/ublthk_ls[n])>=1.0) ratio=1.0;
        else ratio=pow(((j-0.5)*dely/ublthk_ls[n]),0.1429);
        int_1=int_1+(0.5*dh-ublthk_sidmax*j/jblls)*dely*(1-ratio);
    }
    for(j=1;j<=jblsid;j++){
        if(((j-0.5)*dely/ublthk_sid[n])>=1.0) ratio=1.0;
        else ratio=pow(((j-0.5)*dely/ublthk_sid[n]),0.1429);
        int_1=int_1+(ublthk_sid[n]/ublthk_ls[n])*(dh-ublthk_lsmax*j/jblsid-
                            ublthk_tsmax*j/jblsid)*dely*(1-ratio);
    }
    for(j=1;j<=jblts;j++){
        if(((j-0.5)*dely/ublthk_ts[n])>=1.0) ratio=1.0;
        else ratio=pow(((j-0.5)*dely/ublthk_ts[n]),0.1429);
        int_1=int_1+(ublthk_ts[n]/ublthk_ls[n])*(0.5*dh-ublthk_sidmax*j/jblts)
                            *dely*(1-ratio);
    }

    Twls[n]=Tin+(heatflux*x[n]*2*dh/densin/specheat/uin)/int_1;
    Twsid[n]=(Twls[n]-Tin)*ublthk_sid[n]/ublthk_ls[n]+Tin;
    Twts[n]=(Twls[n]-Tin)*ublthk_ts[n]/ublthk_ls[n]+Tin;

/*  Now that I know the wall temperatures at this step.  Compute the
    1/7th profiles  */

    for(j=0;j<=J;j++){
        if(j*dely<=ublthk_ls[n]) Tls[n][j]=Tin+
                      (1-pow((j*dely/ublthk_ls[n]),0.1429))*(Twls[n]-Tin);
        else  Tls[n][j]=Tin;
        densls[n][j]=Pin/Tls[n][j]/Rgas;
    }
    for(j=0;j<=J;j++){
        if(j*dely<=ublthk_sid[n]) Tsid[n][j]=Tin+
                  (1-pow((j*dely/ublthk_sid[n]),0.1429))*(Twsid[n]-Tin);
        else  Tsid[n][j]=Tin;
        denssid[n][j]=Pin/Tsid[n][j]/Rgas;
    }
    for(j=0;j<=J;j++){
        if(j*dely<=ublthk_ts[n]) Tts[n][j]=Tin+
                      (1-pow((j*dely/ublthk_ts[n]),0.1429))*(Twts[n]-Tin);
        else  Tts[n][j]=Tin;
        densts[n][j]=Pin/Tts[n][j]/Rgas;
    }

    return(1);
}
```

```
/*********************************************************/
/*******************************************************/
/******  This subroutine computes the turbulent boundary layer
   profile for the leading wall.  */
float lsturbbl(float blp)
{   int j,g;

    uls[n][0]=0.0;
    for(j=1;j<=J;j++){
        if(j*dely<=ublthk_ls[n]) uls[n][j]=uinftbl[n]*pow((j*dely/ublthk_ls[n]),1.0/blp);
        else uls[n][j]=uinftbl[n];
    }
    uavgls[n]=(blp/(blp+1.0))*uinftbl[n];

    for(j=0;j<=J;j++){
        if(j*dely<=ublthk_ls[n]){
            if(thermalbc==0){
                Tls[n][j]=Tin+(1-pow((j*dely/ublthk_ls[n]),1.0/blp))*(Twls[n]-Tin);
                Tavgls[n]=(Twls[n]+Tin*blp)/(1.0+blp);
            }
            else if(thermalbc==2){
                delQsum1=0.0;
                for(g=1;g<=n;g++) delQsum1=delQsum1+delQlsin[g];
                delQsum3=0.0;
                for(g=1;g<=n;g++) delQsum3=delQsum3+delQcore[g];
                Tavgls[n]=Tin+(qwls*0.5*dh*x[n]+delQsum1-delQsum3)/
                          (specheat*uavgls[n]*ublthk_ls[n]
                          *(0.5*dh-0.5*ublthk_sid[n])*densin);
                nTls=(Twls[n]-Tavgls[n])/(Tavgls[n]-Tin);
                Tls[n][j]=Tin+
                      (1-pow((j*dely/ublthk_ls[n]),1.0/nTls))*(Twls[n]-Tin);
            }
            else Tls[n][j]=Tin;
        }
        else{
            Tls[n][j]=Tin;
        }
        densls[n][j]=Pin/Tls[n][j]/Rgas;
    }

    return(2);
}

/*********************************************************/
/*******************************************************/
/******  This subroutine computes the turbulent boundary layer
   profile for the trailing wall.  */
float tsturbbl(float blp)
{   int j,g;

    uts[n][0]=0.0;
    for(j=1;j<=J;j++){
        if(j*dely<=ublthk_ts[n]) uts[n][j]=uinftbl[n]*pow((j*dely/ublthk_ts[n]),1.0/blp);
        else uts[n][j]=uinftbl[n];
    }
    uavgts[n]=(blp/(blp+1.0))*uinftbl[n];

    for(j=0;j<=J;j++){
        if(j*dely<=ublthk_ts[n]){
            if(thermalbc==0){
                Tts[n][j]=Tin+(1-pow((j*dely/ublthk_ts[n]),1.0/blp))*(Twts[n]-Tin);
                Tavgts[n]=(Twts[n]+Tin*blp)/(1.0+blp);
            }
            else if(thermalbc==2){
                delQsum2=0.0;
                for(g=1;g<=n;g++) delQsum2=delQsum2+delQtsout[g];
                delQsum3=0.0;
                for(g=1;g<=n;g++) delQsum3=delQsum3+delQcore[g];
                Tavgts[n]=Tin+(qwts*0.5*dh*x[n]-delQsum2+delQsum3)/
                          (specheat*uavgts[n]*ublthk_ts[n]
                          *(0.5*dh-0.5*ublthk_sid[n])*densin);
                nTts=(Twts[n]-Tavgts[n])/(Tavgts[n]-Tin);
                Tts[n][j]=Tin+
                      (1-pow((j*dely/ublthk_ts[n]),1.0/nTts))*(Twts[n]-Tin);
            }
            else Tts[n][j]=Tin;
        }
        else{
            Tts[n][j]=Tin;
        }
        densts[n][j]=Pin/Tts[n][j]/Rgas;
    }

    return(3);
}

/******************************************************/
/*****************************************************/
/******  This subroutine computes the turbulent boundary layer
   profile for the side wall.  */
float sidturbbl(float blp)
{   int j,g;

    uavgsid[n]=0.0;
    usid[n][0]=0.0;
    for(j=1;j<=J;j++){
        if(j*dely<=ublthk_sid[n]){
            usid[n][j]=uinftbl[n]*pow((j*dely/ublthk_sid[n]),1.0/blp);
            uavgsid[n]=uavgsid[n]+dely*0.5*(usid[n][j]+usid[n][j-1]+ucor[n][j]+ucor[n][j-1]);
            g=j;
        }
        else usid[n][j]=uinftbl[n];
    }
    uavgsid[n]=uavgsid[n]/(g*dely);

    for(j=0;j<=J;j++){
        if(j*dely<=ublthk_sid[n]){
            if(thermalbc==0){
                Tsid[n][j]=Tin+(1-pow((j*dely/ublthk_sid[n]),1.0/blp))*(Twsid[n]-Tin);
                Tavgsid[n]=(Twsid[n]+Tin*blp)/(1.0+blp);
            }
            else if(thermalbc==2){
                Tavgsid[n]=Tin+(qwsid*dh*x[n]+delQsum2-delQsum1)/
                          (specheat*uavgsid[n]*ublthk_sid[n]
                          *(dh-0. 5*ublthk_ls[n]-0.5*ublthk_ts[n])*densin);
               nTsid=(Twsid[n]-Tavgsid[n])/(Tavgsid[n]-Tin);
                Tsid[n][j]=Tin+
                      (1-pow((j*dely/ublthk_sid[n]),1.0/nTsid))*(Twsid[n]-Tin);
            }
            else Tsid[n][j]=Tin;
        }
        else{
            Tsid[n][j]=Tin;
        }
        denssid[n][j]=Pin/Tsid[n][j]/Rgas;
    }

    return(4);
}

/******************************************************/
/*****************************************************/
/******  This subroutine computes the coefficient to the
  momentum thickness equation.  */
float blcoeff(float p, float u)
{   float c;

    c=pow((p+3.0)/(p+1.0),(p+1.0)/(p+3.0));
    c=c*pow(kvisc,2.0/(p+3.0));
    c=c*pow((pow((p/(p+1.0)-p/(p+2.0)),1/p)/8.75),2.0*p/(p+3.0));
    c=c/pow(u,(3.0*p+2.0)/p);

    return(c);
}
```

Appendix B: Variable Resistivity Code

```
/*  Title:  jpbqvar4.c
    Author: Jeffrey Bons
    Date:   17 Apr 97   */

/*   Calculate Effect of variable resistivity on local heat generation
        Assumes each side is isolated electically with a boundary condition
     of zero lateral potential gradient at the side edges.  Also assumes
           a thickness much less than the length & width (2-D)
                uses values for ITO from 20 Mar 97 run */
/**/
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int side, JJ, J, I, j, i, MM, n, mid;
long int sweep, intvl, cntr;
static double conv, firstRes, Ressum, pi, len, wid, Vo, Vf, Resnorm, Tavg,
                resavg, Qavg, res0C, tcr, thick, Tsum, netcurr, netres, Resside,
                force[18][18], pot[18][18], Resid[18][18], res[18][18],
                dpotdy[18][18], dpotdx[18][18], qloc[18][18], temp[18][18];
FILE *fp;

int jacob(void);

main()
{   fp=fopen("jpbres3.dat","w+");

/*  Input Run Sequence Parameters  */

    printf("Enter # of cells, JJ, in lateral (x) direction:");
    scanf("%d", &JJ);
    printf("Enter # of cells, I, in radial (y) direction:");
    scanf("%d", &I);
    printf("Enter test section side for calculation:");
    printf("0=Leading Side, 1=Side, 2=Trailing Side:");
    scanf("%d", &side);
    printf("Enter # of Jacobi sweeps in iteration:");
    scanf("%ld", &sweep);
    printf("Printing interval, intvl:");
    scanf("%ld", &intvl);

    printf("\nJNodes: %d\tINodes: %d\tSide: %d\tNsweeps: %ld",JJ,I,side,sweep);
    fprintf(fp,"\nJNodes: %d\tINodes: %d\tSide: %d\tNsweeps: %ld",JJ,I,side,sweep);

/*  To use the Boundary Condition of zero gradient, need to use the
    fictitious nodes j = -1, JJ+1 outside the physical domain.        So define */

    J=JJ+2;

/*  Set initial values and constants  */

    conv=100000.0;
    cntr=0;
    pi=3.141592654;

/*  The spatial grid dimensions for one side of the test section, [meters]*/

    thick=0.00000085;
    len=0.09;
    wid=0.01;

/*  Inlet and exit voltage potential levels , [volts] */

    Vo=0.0;
    Vf=23.5;

/*  Initialize the forcing array  */
/*  Begin iteration with linear lengthwise potential gradient  */
/*  and no lateral potential gradients */

    for(j=0;j<=J;j++){
        for(i=0;i<=I;i++){
            force[j][i]=0.0;
            pot[j][i]=Vo+(i*len/I)*(Vf-Vo)/len;
/*          pot[j][i]=Vo+(Vf-Vo)*pow((i*len/I)/len,2); for linear T*/
            if(i==I) force[j][i]=Vf;
            if(i==0) force[j][i]=Vo;
        }
    }

/*  Call the subroutine to perform the iteration      */

    jacob();

/*  Print Residue at Last Iteration */

    printf("\nCounter\tRessum\t\tFirstRes\tResnorm");
    fprintf(fp,"\nCounter\tRessum\t\tFirstRes\tResnorm");
    printf("\n%ld\t%.9f\t%.9f\t%.9f", cntr-1, Ressum,
                        firstRes, Resnorm);
    fprintf(fp,"\n%ld\t%.9f\t%.9f\t%.9f", cntr-1, Ressum,
                        firstRes, Resnorm);

/*  Calculate average resistance and heat flux*/

    Tavg=Tsum/((JJ+1)*(I+1));
    resavg=res0C*(1+tcr*(Tavg-0.0));
    Resside=resavg*len/(wid*thick);
    Qavg=(pow((Vf-Vo),2)/(wid*len))*(1/Resside);
    printf("\nAvg Temp\tAvg Resist\tSide Res\tAvg Heat Flux [W/m2]");
    fprintf(fp,"\nAvg Temp\tAvg Resist\tSide Res\tAvg Heat Flux [W/m2]");
    printf("\n%.2f\t\t%.9f\t%.2f\t%.2f",Tavg,resavg,Resside,Qavg);
    fprintf(fp,"\n%.2f\t\t%.9f\t%.2f\t%.2f",Tavg,resavg,Resside,Qavg);

/*  Calculate local heat flux  */

    fprintf(fp,"\nMatrix of surface heat flux variation [qloc/qavg-1]%");
    for(i=0;i<=I;i++){
        fprintf(fp,"\n");
        for(j=1;j<J;j++){
            if(i==0){
                dpotdy[j][i]=(pot[j][i+1]-pot[j][i])/(len/I);
                dpotdx[j][i]=(pot[j+1][i]-pot[j-1][i])/(2*wid/(J-2));
            }
            else if(i==I){
                dpotdy[j][i]=(pot[j][i]-pot[j][i-1])/(len/I);
                dpotdx[j][i]=(pot[j+1][i]-pot[j-1][i])/(2*wid/(J-2));
            }
            else{
                dpotdy[j][i]=(pot[j][i+1]-pot[j][i-1])/(2*len/I);
                dpotdx[j][i]=(pot[j+1][i]-pot[j-1][i])/(2*wid/(J-2));
            }
            qloc[j][i]=(pow(dpotdy[j][i],2)+pow(dpotdx[j][i],2))/res[j][i];
            fprintf(fp,"%.2f  ",100*(qloc[j][i]*thick/Qavg-1.0));
        }
    }

/*  Calculate bulk side resistance by first computing the net
    current through the side at mid-span */

    mid=abs(I/2);
    netcurr=0.0;
    for(j=1;j<J;j++){
        if(j==1 | j==(J-1)){
            netcurr=netcurr+(wid/JJ/2)*(dpotdy[j][mid])/res[j][mid];
        }
        else{
            netcurr=netcurr+(wid/JJ)*(dpotdy[j][mid])/res[j][mid];
        }
    }
    netcurr=netcurr*thick;
    netres=(Vf-Vo)/netcurr;
    fprintf(fp,"\nCurrent through side = %.4f Amps",netcurr);
    fprintf(fp,"\nSide Resistance= %.4f Ohms",netres);

    fprintf(fp,"\n");

    fclose(fp);
    return(0);
}


/*  Subroutine to perform Jacobi sweeps */
jacob()
{   int p, i, j, mec;
    long int n;
    static double Tin, delTy, delTx, deltax, deltay, dx2, dy2, k,
                  dresdx[18][18], dresdy[18][18], dumpot[18][18];

 /*  Set up temperature/resistivity functions    */

     res0C=0.000005672; /* [ohm-meters]  for side 2 */
     tcr=0.00031;       /* [ohm-meters per degree C]  */
     Tin=138.0;         /* [degC] wall temp at inlet  */
     delTy=29.0;        /* [degC] max temp variation end to end       */
     delTx=40.0;        /* [degC] max temp variation around perimeter */
     delTy=29.0;        /* [degC] max temp variation end to end       */
     delTx=40.0;        /* [degC] max temp variation around perimeter */

 /*  Set up grid dimensions at this level   */

     deltax=wid/JJ;     /* Due to added side nodes     */
     deltay=len/I;
     dx2=deltax*deltax;
     dy2=deltay*deltay;
     k=0.5*(dy2*dx2/(dy2+dx2));

 /*  Compute local values of temperature, resistivity, and resistivity gradients*/

     Tsum=0;
     printf("\nTemp Matrix Computed\n");
     fprintf(fp,"\nLocal Temp Matrix");

     p=side*(J-2);

     for(i=0;i<=I;i++){
         fprintf(fp,"\n");
         for(j=p;j<=(p+J-2);j++){
             temp[j-p+1][i]=Tin+2.0*delTy*i*deltay/(0.75*len)-delTy*
                 pow(i*deltay/(0.75*len),2)+0.5*delTx*sin(pi*i*deltay/len)
                 *cos(2.0*pi*(j*deltax-wid/2.0)/(4.0*wid));
             res[j-p+1][i]=res0C*(1+tcr*(temp[j-p+1][i]-0.0));
             dresdx[j-p+1][i]=-res0C*tcr*delTx*pi*sin(pi*i*deltay/len)
                 *sin(2.0*pi*(j*deltax-wid/2.0)/(4.0*wid))/(4.0*wid);
             dresdy[j-p+1][i]=res0C*tcr*(2.0*delTy/(0.75*len)-2.0*delTy*
                 i*deltay/pow(0.75*len,2)+0.5*delTx*pi*cos(pi*i*deltay/len)
                 *cos(2.0*pi*(j*deltax-wid/2.0)/(4.0*wid))/len);

 /*   for linear Temp gradient    */

 /*          temp[j-p+1][i]=Tin+delTy*i*deltay/len;
             res[j-p+1][i]=res0C*(1+tcr*(temp[j-p+1][i]-0.0));
             dresdx[j-p+1][i]=0.0;
             dresdy[j-p+1][i]=res0C*tcr*delTy/len;  */

             Tsum=Tsum+temp[j-p+1][i];
             fprintf(fp,"%.2f  ",(temp[j-p+1][i]+273));
         }
     }

     fprintf(fp,"\nLocal Resist Matrix");
     for(i=0;i<=I;i++){
         fprintf(fp,"\n");
         for(j=p;j<=(p+J-2);j++) fprintf(fp,"%.3e  ",res[j-p+1][i]);
     }
     fprintf(fp,"\nLocal dresdx Matrix");
     for(i=0;i<=I;i++){
         fprintf(fp,"\n");
         for(j=p;j<=(p+J-2);j++) fprintf(fp,"%.3e  ",dresdx[j-p+1][i]);
     }
     fprintf(fp,"\nLocal dresdy Matrix");
     for(i=0;i<=I;i++){
         fprintf(fp,"\n");
         for(j=p;j<=(p+J-2);j++) fprintf(fp,"%.3e  ",dresdy[j-p+1][i]);
     }

MM=(J+1)*(I+1)-1;  /* Matrix size at this level */

/*  Perform Jacobi sweeps  */

    for(n=1;n<=sweep;n++){
        for(j=0;j<=J;j++){
            for(i=0;i<=I;i++){
                dumpot[j][i]=pot[j][i];
            }
        }
        for(j=0;j<=J;j++){
            for(i=0;i<=I;i++){
                if(i==0) pot[j][i]=force[j][i];
                else if(i==I) pot[j][i]=force[j][i];
                else if(j==0) pot[j][i]=dumpot[j+2][i]+force[j][i];
                else if(j==J) pot[j][i]=dumpot[j-2][i]+force[j][i];
                else pot[j][i]=(k/res[j][i])*(
                    (res[j][i]/dx2-dresdx[j][i]*0.5/deltax)*dumpot[j+1][i]
                    +(res[j][i]/dx2+dresdx[j][i]*0.5/deltax)*dumpot[j-1][i]
                    +(res[j][i]/dy2-dresdy[j][i]*0.5/deltay)*dumpot[j][i+1]
                    +(res[j][i]/dy2+dresdy[j][i]*0.5/deltay)*dumpot[j][i-1])
                    +force[j][i];
            }
        }

 /*  Compute the residue at this iteration, tally the residue sum,
     and compare it to the residue from the first iteration */

        Ressum=0.0;
        for(j=0;j<=J;j++){
            for(i=0;i<=I;i++){
              if(i==0) Resid[j][i]=0.0;
              else if(i==I) Resid[j][i]=0.0;
              else{
                if(j==0) Resid[j][i]= -pot[j][i]+pot[j+2][i]+force[j][i];
                else if(j==J) Resid[j][i]= -pot[j][i]+pot[j-2][i]+force[j][i];
                else Resid[j][i]= -pot[j][i] + (k/res[j][i])*(
                    (res[j][i]/dx2-dresdx[j][i]*0.5/deltax)*pot[j+1][i]
                    +(res[j][i]/dx2+dresdx[j][i]*0.5/deltax)*pot[j-1][i]
                    +(res[j][i]/dy2-dresdy[j][i]*0.5/deltay)*pot[j][i+1]
                    +(res[j][i]/dy2+dresdy[j][i]*0.5/deltay)*pot[j][i-1])
                    +force[j][i];
              }
              Ressum=Ressum+fabs(Resid[j][i]);
            }
        }


        if(cntr==0){
            firstRes=Ressum;
            mec=cntr;
        }
        Resnorm=log10(Ressum/firstRes);
        if(mec==intvl){
            mec=mec-intvl;
            printf("\ncntr%ld\t%.9f\t%.9f\t%.9f", cntr, Ressum,
                                       firstRes, Resnorm);
            fprintf(fp,"\ncntr%ld\t%.9f\t%.9f\t%.9f", cntr, Ressum,
                                       firstRes, Resnorm);
        }
        cntr=cntr+1;
        mec=mec+1;
        if(firstRes/Ressum>conv) break;
    }

    return(1);
}
```

Appendix C: PIV Processing Code

```
%
%       Title:  jpbpiv3.m
%       Author:  Jeffrey Bons
%       Parent program: none (This is the TOP LEVEL)
%       Child subroutine(s): too many to list...
%       Date: 25 Mar 1997
%
% This program creates a Matlab window (graphical user interface) with menus
% that allow PIV images to be read in, interrogation boxes to be
% defined and the correlated using a cross-correlation technique.
% Finally, the resulting vector field can be plotted.

% First a figure window is opened and titled.  This window will then
% be filled with all of the necessary menus.

MainWin=1;
figure(MainWin);
set(MainWin, 'resize','on','name','JPBPIV3','menubar','none')

% All of the other windows used are named and defaulted to =0

ImageWin=0;
LTSWin=0;
HistWin=0;
PlotWin=0;
VectWin=0;
CorrWin=0;

% Initialize some of the interrogation box parameters here so that
% they don't have to be re-defined with each new image.  This is
% especially helpful with stationary images, as the window is
% usually constant.
% The first 4 are used in CCbox.m the last 3 are used in CCprepplot.m

ulxrect=[];
ulyrect=[];
lrxrect=[];
lryrect=[];
alpha=[];
xts=[];
yts=[];

%----------------------------------
%File menu and its submenus:
% The file menu permits access to 4 subroutines which perform
% file I/O type operations.  The subroutines are: CCloadim, CCsaveim,
% printimjb, & CCcloseim.  There is also a selection that allows the
% user to do a save/print/and close all in a row.
% The last menu item allows an exit from jpbpiv3.
%-----------------------------------
File_menu=uimenu(MainWin,'Label','File    ');

Open_submenu=uimenu(File_menu,'Label','Load .tif File','Callback','fprintf(''Loading %s.tif
...\n'',filename);CCloadim;fprintf(''Done \n\n'') ');

Save_submenu=uimenu(File_menu,'Label','Save All Files', 'Callback',
'fprintf(''Saving...\n'');CCsaveim;fprintf(''Done \n\n'') ');

Print_submenu=uimenu(File_menu,'Label','Print Figures','Callback','printimjb;fprintf(''Done \n\n'')
');

SavPrntCls_submenu=uimenu(File_menu,'Label','Save&Print&Close','Callback','fprintf(''Saving...\n'');
CCsaveim;fprintf(''Done Saving\n'');printimjb;fprintf(''Done Printing\n'');CCcloseim;fprintf(''Done
Closing\n'') ');

Close_submenu=uimenu(File_menu,'Label','Close', 'Callback', 'fprintf(''Closing image windows and
deleting variables ...\n'');CCcloseim;fprintf(''Done \n\n'')  ');

Exit_submenu=uimenu(File_menu,'Label','Exit', 'Callback', 'fprintf(''Quitting JPBPIV3 ...\n'');close
all;clear;clear global;fprintf(''Done \n\n'')  ');


%----------------------------------
% Display data menu and its submenus:
% The first menu item allows the user to replot the PIV
% image at any time during the processing session.
% CCspotrem.m is used to remove regions of glare from the image
% so that they do not bias the cross-correlator.
% CCbox.m allows the user to define the region which is then
% divided up into interrogation boxes and displayedin LTSWin.
% It helps to have a color monitor!!!!
%----------------------------------
Display_menu=uimenu(MainWin,'Label','Show/Box Image');

ShowPrs_submenu=uimenu(Display_menu,'Label','Show Particles','Callback', 'if (ImageWin==0)
ImageWin=figure;else figure(ImageWin); end;set(ImageWin,''name'',filename);clg;set(gca,
''position'',[0 0 1 1]);imshow(imdata,mapimdata);' );

RemSpot_submenu=uimenu(Display_menu,'Label','Remove Glare Spots','Callback','
CCspotrem;fprintf(''Done\n'');');

CCBoxes_submenu=uimenu(Display_menu,'Label','Define CC Boxes','Callback','
CCbox;fprintf(''Done\n'');');

%----------------------------------
% Alignment menu is for rotating piv images only.
% Double exposures of the rotating alignment tool
% are interrogated to determine the x,y pixel
% dimensions and then the center of rotation in
% pixel coordinates.
% pixmag.m is used to adjust these alignment values for
% the magnification of the camera as I refocus on different
% planes through the test section thickness.
%----------------------------------
Alignment_menu=uimenu(MainWin,'Label','Rotating Align   ');

Calcorr_submenu=uimenu(Alignment_menu,'Label','Correlate Algn Tool','Callback', 'fprintf(''Correlate
sections of alignment tool image...\n'');corrimagjb2;fprintf(''Done with cross, now do
horizontal\n'');corrimagjb2;fprintf(''Done with horizontal\n'')');

Calcpixdim_submenu=uimenu(Alignment_menu,'Label','Calc pixel wid/ht','Callback', 'fprintf(''Sending
correlated data to compute pixel dimensions...\n'');xyresol;fprintf(''Done \n'')');

Crclcent_submenu=uimenu(Alignment_menu,'Label','Calc center of rot','Callback', 'fprintf(''Compute
x,y center of rotation...\n'');crclcntr2;fprintf(''Done \n'')');

PixANDcent_submenu=uimenu(Alignment_menu,'Label','Calc pixdim & cntrrot','Callback',
'fprintf(''Sending correlated data to compute pixel dimensions...\n'');xyresol;fprintf(''Done
\n'');fprintf(''Compute x,y center of rotation...\n'');crclcntr2;fprintf(''Done \n'')');

AdjMag_submenu=uimenu(Alignment_menu,'Label','Adjust Magn for z/d','Callback','fprintf(''Initiating
magnification adjustment...\n'');pixmag;fprintf(''Done\n'')');

RpsDet_submenu=uimenu(Alignment_menu,'Label','Determine rps','Callback','fprintf(''Initiating rps
determination...\n'');rpsdet;fprintf(''Done\n'')');

%----------------------------------
% Rotate menu is for all processing and plotting functions.
% CCcorrbox3.m - correlates boxes previously defined
% CCrecorrbox4.m - recorrelates boxes passes up by CCcorrbox.m
% These can all be done in sequence without user prompt
% CCredovect2.m - after cleanup can redo a single "suspect" vector if desired.
% CCprepplot.m - transfers to TS ref frame
% CCplot3.m - plots in TS frame
% CCprepplot and CCplot3 can also be done in sequence
% CCvectclr.m - creates vector plots with color spectrum
% to signify various levels of vector magnitude.
%----------------------------------
Rotate_menu=uimenu(MainWin,'Label','Rotating PIV   ');

Corrbox_submenu=uimenu(Rotate_menu,'Label','Correlate boxes','Callback', 'fprintf(''Correlating
boxes...\n'');CCcorrbox3;fprintf(''Done \n'')');

Recorrbox_submenu=uimenu(Rotate_menu,'Label','Recorrelate boxes','Callback',
'fprintf(''Recorrelating boxes...\n'');CCrecorrbox4;fprintf(''Done \n'')');

Redo1_submenu=uimenu(Rotate_menu,'Label','Redo 1Vector','Callback',
'fprintf(''Redoing...\n'');CCredovect2;fprintf(''Done \n'')');

Both_submenu=uimenu(Rotate_menu,'Label','Auto Corr&ReCorr','Callback', 'fprintf(''Correlating
boxes...\n'');tic;CCcorrbox3;fprintf(''Done\n'');fprintf(''Recorrelating
boxes...\n'');CCrecorrbox4;fprintf(''Done\n'');toc;fprintf(''All Done\n'')');

Preplot_submenu=uimenu(Rotate_menu,'Label','Prep for Plot','Callback',' fprintf(''Prep for
plotting...\n'');CCprepplot;fprintf(''Done\n\n'');' );
```

```
Plot_submenu=uimenu(Rotate_menu,'Label','Plot CC Data','Callback','
fprintf(''Plotting...\n'');CCplot3;fprintf(''Done\n\n'');' );

ClrPlot_submenu=uimenu(Rotate_menu,'Label','Color Vector Plot','Callback','
fprintf(''Plotting...\n'');CCvectclr;fprintf(''Done\n\n'');' );

Plot2_submenu=uimenu(Rotate_menu,'Label','Auto Prep&Plot','Callback',' fprintf(''Prep for
plotting...\n'');CCprepplot;fprintf(''Plotting...\n'');CCplot3;fprintf(''Done\n\n'');' );



%-----------------------------------
%-----------------------------------
%-----------------------------------


%-----------------------------------
% Boxes for entering the path & file name
% (Be sure to only enter the filename prefix, not
% the .tif suffix, or the saved files will not work).
%-----------------------------------
uicontrol(MainWin,'Style','Frame','BackgroundColor',[1 1 1],'Position',[1 280 510 55 ]);

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Path
name:','Position',[ 5 308 110 25 ]);

PathName_edit=uicontrol(MainWin,'Style','edit','HorizontalAlignment','left','BackgroundColor',[ 1 1
1],'String','','Position',[ 115 308 390 25  ], 'Callback', 'pathname=get(PathName_edit,''string'');'
);

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','File name
prefix:','Position',[ 5 283 110 25 ]);

FileName_edit=uicontrol(MainWin,'Style','edit','HorizontalAlignment','left','BackgroundColor',[ 1 1
1],'String','','Position',[ 115 283 390 25], 'Callback',
'filename=get(FileName_edit,''string'');');


%-----------------------------------
% Boxes for entering Re#, Rot#, and z/d position.
%-----------------------------------
uicontrol(MainWin,'Style','Frame','BackgroundColor',[1 1 1],'Position',[1 252 510 26 ]);
uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Run Parameters:
','Position',[ 5 255 140 20 ]);
uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Re#:','Position',[ 160
255 40 20 ]);
ImAttrib_edit60=uicontrol(MainWin,'Style','edit','HorizontalAlignment','left','BackgroundColor',[ 1
1 1],'String','','Position',[ 205 255 60 20  ], 'Callback', 'Renum=get(ImAttrib_edit60,''string'');'
);
uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Rot#:','Position',[
280 255 40 20 ]);
ImAttrib_edit61=uicontrol(MainWin,'Style','edit','HorizontalAlignment','left','BackgroundColor',[ 1
1 1],'String','','Position',[ 325 255 60 20  ], 'Callback',
'Rotnum=get(ImAttrib_edit61,''string'');' );
uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','z/d:','Position',[ 400
255 40 20 ]);
ImAttrib_edit62=uicontrol(MainWin,'Style','edit','HorizontalAlignment','left','BackgroundColor',[ 1
1 1],'String','','Position',[ 445 255 60 20  ], 'Callback',
'zoverdh=get(ImAttrib_edit62,''string'');' );


%-----------------------------------
% Boxes for entering the pairing parameters used by
% CCcorrbox.m   Parameters are self-explanatory.
%-----------------------------------
uicontrol(MainWin,'Style','Frame','BackgroundColor',[1 1 1],'Position',[1 75 250 165 ]);

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','PIV
Parameters','Position',[ 5 218 240 20 ]);

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Pixel Width
[um]','Position',[ 5 198 130 20 ]);
ImAttrib_edit14=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 135 198 110 20 ], 'Callback',
'pixel_width=str2num(get(ImAttrib_edit14,''string''));');

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Pixel Height
[um]','Position',[ 5 178 130 20 ]);
ImAttrib_edit15=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 135 178 110 20 ], 'Callback',
'pixel_height=str2num(get(ImAttrib_edit15,''string''));');
```

```
uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Pulse Interval
[us]','Position',[ 5 158 130 20 ]);

ImAttrib_edit22=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 135 158 110 20 ], 'Callback',
'pulse_dt=str2num(get(ImAttrib_edit22,''string''));');


uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Min Velocity
[m/s]','Position',[ 5 138 130 20 ]);

ImAttrib_edit23=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 135 138 110 20 ], 'Callback',
'min_velocity=str2num(get(ImAttrib_edit23,''string''));');


uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Max Velocity
[m/s]','Position',[ 5 118 130 20 ]);

ImAttrib_edit24=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 135 118 110 20 ], 'Callback',
'max_velocity=str2num(get(ImAttrib_edit24,''string''));');


uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Min Angle
[deg]','Position',[ 5 98 130 20 ]);

ImAttrib_edit25=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 135 98 110 20], 'Callback',
'min_angle=str2num(get(ImAttrib_edit25,''string''));');


uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Max angle
[deg]','Position',[ 5 78 130 20 ]);

ImAttrib_edit26=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 135 78 110 20 ], 'Callback',
'max_angle=str2num(get(ImAttrib_edit26,''string''));');


%-----------------------------------

% Boxes for entering the interrogation box dimensions used by
% CCbox.m:
% box_width - width of boxes used for cross-correlation
% box_height - height of boxes used for cross-correlation
% (These were designed for just odd integers, though it may work
% fine with even integers too.)
%-----------------------------------

uicontrol(MainWin,'Style','Frame','BackgroundColor',[1 1 1],'Position',[1 5 250 65 ]);

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Box Dimensions (odd
pixels only)','Position',[ 5 48 240 20 ]);


uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Box Width
[pix]','Position',[ 5 28 130 20 ]);

ImAttrib_edit31=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[135 28 110 20 ], 'Callback',
'box_width=str2num(get(ImAttrib_edit31,''string''));');


uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Box Height
[pix]','Position',[ 5 8 130 20 ]);

ImAttrib_edit32=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[135 8 110 20 ], 'Callback',
'box_height=str2num(get(ImAttrib_edit32,''string''));');
```

```
%----------------------------------

% Boxes for entering the cleanup parameter used
% by CCcleanup2.m.  'Reduce' is the amount the min/max
% velocity and angle ranges should be reduced on the
% cleanup pass through the boxes.
%----------------------------------
uicontrol(MainWin,'Style','Frame','BackgroundColor',[1 1 1],'Position',[261 205 250 45 ]);

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Parameters for
Cleanup','Position',[ 265 228 240 20 ]);


uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Recorr Threshold
','Position',[ 265 208 130 20 ]);

ImAttrib_edit64=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 395 208 110 20 ], 'Callback',
'reduce=str2num(get(ImAttrib_edit64,''string''));');



%----------------------------------
% Boxes for entering the parameters used by
% rotating piv.
% Parameters are self-explanatory.
%----------------------------------
uicontrol(MainWin,'Style','Frame','BackgroundColor',[1 1 1],'Position',[261 55 250 145 ]);

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Rotating
Parameters','Position',[ 265 178 240 20 ]);

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Rotation freq
[rps]','Position',[ 265 158 130 20 ]);
ImAttrib_edit41=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 395 158 110 20 ], 'Callback',
'rotfreq_rps=str2num(get(ImAttrib_edit41,''string''));');

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Cntr of Rot-X
pixel','Position',[ 265 138 130 20 ]);
ImAttrib_edit42=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 395 138 110 20 ], 'Callback',
'xcntr_pix=str2num(get(ImAttrib_edit42,''string''));');

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Cntr of Rot-Y
pixel','Position',[ 265 118 130 20 ]);
ImAttrib_edit43=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 395 118 110 20 ], 'Callback',
'ycntr_pix=str2num(get(ImAttrib_edit43,''string''));');

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Algn cross
rad[um]','Position',[ 265 98 130 20 ]);
ImAttrib_edit44=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 395 98 110 20 ], 'Callback',
'rad_cross=str2num(get(ImAttrib_edit44,''string''));');

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Algn wire
dist[um]','Position',[ 265 78 130 20 ]);
ImAttrib_edit45=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 395 78 110 20 ], 'Callback',
'dist_hwire=str2num(get(ImAttrib_edit45,''string''));');

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','CamTrav Mot
(um)','Position',[ 265 58 130 20 ]);

ImAttrib_edit46=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 395 58 110 20 ], 'Callback',
'cammotion=str2num(get(ImAttrib_edit46,''string''));');




%----------------------------------

% Boxes for entering the scaling factor for velocity vectors
%----------------------------------
uicontrol(MainWin,'Style','Frame','BackgroundColor',[1 1 1],'Position',[261 5 250 45 ]);
uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Display
Parameters','Position',[ 265 28 240 20 ]);

uicontrol(MainWin,'Style','text','BackgroundColor',[0.85 0.85 0.85],'String','Scale of Vel
Vect','Position',[ 265 8 130 20 ]);

ImAttrib_edit51=uicontrol(MainWin,'Style','edit','BackgroundColor',[1 1
1],'String','','HorizontalAlignment','left','Position',[ 395 8 110 20 ], 'Callback',
'scale=str2num(get(ImAttrib_edit51,''string''));');



%----------------------------------



%
%       Title:  CCloadim.m
%       Author:  Jeffrey Bons
%       Parent program: jpbpiv3.m
%       Child subroutine(s): tiffread.m, imshow.m
%       Date: 4 Mar 1997
%
% Essentially, CCloadim.m loads in the image which must
% be in the TIF format (tagged image file).  If the file
% cannot be read, try resaving it using "xv".
% This somehow decompresses .tif files into a format
% readable by Matlab's "tiffread" toolbox.  After loading
% in the image, any existing files are also read in.
%
% tiffread.m is one of Matlab's Image Processing Toolbox
% programs.  It basically produces a 2D matrix of y-by-x
% pixels (imdata), each entry of which is a number from
% 8 to 256 (8 bit).  The numbers 8 to 256 correspond to
% numbers in the vector mapimdata, which produces the
% corresponding color in RGB format.  256 is bright white
% and 8 is dark black for the monochromatic images
% which are typically used here.

[imdata,mapimdata]=tiffread([pathname,'/',filename]);

% Once the image is loaded, it is displayed using
% imshow.m (another IP toolbox) in the figure entitled
% ImageWin.

if (ImageWin==0)
  ImageWin=figure;
else figure(ImageWin);
end;

set(ImageWin,'name',filename);
clg;
set(gca, 'position',[0 0 1 1]);
imshow(imdata, mapimdata);

% The dimensions of the pixel space are computed, and
% the mean intensity value is calculated.

[msizeimdata ,nsizeimdata]=size(imdata);
absmean=mean(mean(imdata));
fprintf('Mean gray value:  %f\n\n',absmean);

% Default values set for parameters of interest

rotfreq_rps=[];
xcntr_pix=[];
ycntr_pix=[];
pixel_width=[];
pixel_height=[];
pulse_dt=[];
min_velocity=[];
max_velocity=[];
min_angle=[];
max_angle=[];
box_width=[];
box_height=[];
cammotion=[];
reduce=[];
scale=[];
Renum=[];
Rotnum=[];
```

```
zoverdh=[];

xbox=[];
ybox=[];

difx=[];
dify=[];
ang=[];
vel=[];
goodbox=[];

difeps=[];
difdel=[];
epsbox=[];
delbox=[];
TSang=[];
TSvel=[];

corrxy=[];
lftcrossx=[];
lftcrossy=[];
vfact=[];
hfact=[];
ratfact=[];
rotcentx=[];
rotcenty=[];

% If there's already a .mat file, it is loaded in.  Before
% loading in an image, it's wise to copy an old .mat file
% into a .mat file of the current filename to save time.

fid=fopen([pathname,'/',filename,'.mat'], 'r');
if fid>=0
    fclose(fid);
    load ([pathname,'/',filename,'.mat']);
    fprintf('Loading *.mat file ...\n');

    set(ImAttrib_edit14,'string',num2str(pixel_width));
    set(ImAttrib_edit15,'string',num2str(pixel_height));
    set(ImAttrib_edit22,'string',num2str(pulse_dt));
    set(ImAttrib_edit23,'string',num2str(min_velocity));
    set(ImAttrib_edit24,'string',num2str(max_velocity));
    set(ImAttrib_edit25,'string',num2str(min_angle));
    set(ImAttrib_edit26,'string',num2str(max_angle));

    set(ImAttrib_edit31,'string',num2str(box_width));
    set(ImAttrib_edit32,'string',num2str(box_height));

    set(ImAttrib_edit64,'string',num2str(reduce));

    set(ImAttrib_edit41,'string',num2str(rotfreq_rps));
    set(ImAttrib_edit42,'string',num2str(xcntr_pix));
    set(ImAttrib_edit43,'string',num2str(ycntr_pix));
    set(ImAttrib_edit44,'string',num2str(rad_cross));
    set(ImAttrib_edit45,'string',num2str(dist_hwire));
    set(ImAttrib_edit46,'string',num2str(cammotion));

    set(ImAttrib_edit51,'string',num2str(scale));

    set(ImAttrib_edit60,'string',num2str(Renum));
    set(ImAttrib_edit61,'string',num2str(Rotnum));
    set(ImAttrib_edit62,'string',num2str(zoverdh));

else
    fprintf('No *.mat file available!! \n');
end

% Before looking for other files, find out if they exist.

cont=input('Search for other related image processing files?  y/n <n>: ','s');
if isempty(cont)
    cont='n';
end

if cont=='y'

% If it exists, read in the .box file containing the
% interrogation box center indices.

    fid=fopen([pathname,'/',filename,'.box'], 'r');
    if fid<0
```

```
        fprintf('No *.box file available!!\n');
    else
        fclose(fid);
        load ([pathname,'/',filename,'.box']);
        eval(['ulxrect=' filename '(1);'])
        eval(['ulyrect=' filename '(2);'])
        eval(['lrxrect=' filename '(3);'])
        eval(['lryrect=' filename '(4);'])
        eval(['limx=' filename '(5);'])
        eval(['limy=' filename '(6);'])
        for k=1:limx
            eval(['xbox(k)=' filename '(6+k);'])
        end
        for j=1:limy
            eval(['ybox(j)=' filename '(6+j+limx);'])
        end
        fprintf('Loading *.box file ...\n');
        eval(['clear ' filename])
    end

% If it exists, read in the .cor file which contains the
% angle and velocity vectors for each box along with the
% status in goodbox.

    fid=fopen([pathname,'/',filename,'.cor'], 'r');
    if fid<0
        fprintf('No *.cor file available!!\n');
    else
        fclose(fid);
        load ([pathname,'/',filename,'.cor']);
        for k=1:length(xbox)
            for j=1:length(ybox)
                eval(['difx(j,k)=' filename '(((k-1)*length(ybox)+j),1);'])
                eval(['dify(j,k)=' filename '(((k-1)*length(ybox)+j),2);'])
                eval(['ang(j,k)=' filename '(((k-1)*length(ybox)+j),3);'])
                eval(['vel(j,k)=' filename '(((k-1)*length(ybox)+j),4);'])
                eval(['goodbox(j,k)=' filename '(((k-1)*length(ybox)+j),5);'])
            end
        end
        fprintf('Loading *.cor file ...\n');
        eval(['clear ' filename])
    end

% If it exists, read in .plt which contains all the data
% used for plotting.

    fid=fopen([pathname,'/',filename,'.plt'], 'r');
    if fid<0
        fprintf('No *.plt file available!!\n');
    else
        fclose(fid);
        load ([pathname,'/',filename,'.plt']);
        eval(['alpha=' filename '(1,1);'])
        eval(['xts=' filename '(1,2);'])
        eval(['yts=' filename '(1,3);'])
        [mdat,ndat]=size(eval(filename));
        eval(['datdum=' filename '(2:mdat,:);'])
        for k=1:length(xbox)
            for j=1:length(ybox)
                difeps(j,k)=datdum(((k-1)*length(ybox)+j),1);
                difdel(j,k)=datdum(((k-1)*length(ybox)+j),2);
                delbox(j,k)=datdum(((k-1)*length(ybox)+j),3);
                epsbox(j,k)=datdum(((k-1)*length(ybox)+j),4);
                TSang(j,k)=datdum(((k-1)*length(ybox)+j),5);
                TSvel(j,k)=datdum(((k-1)*length(ybox)+j),6);
            end
        end
        fprintf('Loading *.plt file ...\n');
        eval(['clear ' filename])
        clear datdum
    end

% Finally, if this is an alignment image, there will be an
% .alg file with the alignment data.

    fid=fopen([pathname,'/',filename,'.alg'], 'r');
    if fid<0
        fprintf('No *.alg file available!!\n');
    else
        fclose(fid);
        load ([pathname,'/',filename,'.alg']);
        [malg,nalg]=size(eval(filename));
        eval(['corrxy=' filename '(1:2,1:2);'])
        if (malg>=3)
            eval(['lftcrossx=' filename '(3,1);'])
            eval(['lftcrossy=' filename '(3,2);'])
        end
        if (malg>=4)
            eval(['vfact=' filename '(4,1);'])
            eval(['hfact=' filename '(4,2);'])
        end
        if (malg>=5)
            eval(['rotcentx=' filename '(5,1);'])
            eval(['rotcenty=' filename '(5,2);'])
        end
        eval(['clear ' filename])
        fprintf('Loading *.alg file ...\n');
    end
end



%
%       Title:  CCsaveim.m
%       Author: Jeffrey Bons
%       Parent program: jpbpiv3.m
%       Child subroutine(s):
%       Date: 14 Nov 96
%
%  This program saves any data generated while running the master
%  program entitled jpbpiv3.m:
%
%       .box    saves box and rectangle coordinates (from CCbox.m)
%       .cor    saves velocity data in image ref frame (from CCcorrbox.m)
%       .plt    saves velocity data in TS ref frame (from CCplot.m)
%       .mat    saves the front panel settings
%       .alg    saves parameters from rotating alignment tool images
%
%  The .mat file is generated automatically.  All the other files are only
%  generated if the corresponding analysis actually took place during the
%  session.

if xbox ~=[]
    fprintf('\nSaving *.box file ... \n')
    eval([ 'fid=fopen(''',pathname,'/',filename,'.box'', ''w''); ' ]);
    fprintf(fid,'%d\n',ulxrect);
    fprintf(fid,'%d\n',ulyrect);
    fprintf(fid,'%d\n',lrxrect);
    fprintf(fid,'%d\n',lryrect);
    fprintf(fid,'%d\n',length(xbox));
    fprintf(fid,'%d\n',length(ybox));
    fprintf(fid,'%d\n',xbox');
    fprintf(fid,'%d\n',ybox');
    fclose(fid);

    if difx~=[]
        fprintf('Saving *.cor file ... \n')
        eval([ 'fid=fopen(''',pathname,'/',filename,'.cor'', ''w''); ' ]);

        dum1=[];
        dum2=[];
        dum3=[];
        dum4=[];
        dum5=[];
        for k=1:length(xbox)
            for j=1:length(ybox)
                dum1=[dum1;difx(j,k)];
                dum2=[dum2;dify(j,k)];
                dum3=[dum3;ang(j,k)];
                dum4=[dum4;vel(j,k)];
                dum5=[dum5;goodbox(j,k)];
            end
        end

        all=[dum1'; dum2'; dum3'; dum4'; dum5'];
        fprintf(fid,'%8.4f %8.4f %8.4f %8.4f %d\n',all);
        fclose(fid);
    end

    if difeps~=[]
        fprintf('Saving *.plt file ... \n')
        eval([ 'fid=fopen(''',pathname,'/',filename,'.plt'', ''w''); ' ]);

        dum1=[];
        dum2=[];
        dum3=[];
        dum4=[];
        dum5=[];
        dum6=[];
        dum1(1)=alpha;
        dum2(1)=xts;
        dum3(1)=yts;
        dum4(1)=0;
        dum5(1)=0;
        dum6(1)=0;
        for k=1:length(xbox)
            for j=1:length(ybox)
                dum1=[dum1;difeps(j,k)];
                dum2=[dum2;difdel(j,k)];
                dum3=[dum3;delbox(j,k)];
                dum4=[dum4;epsbox(j,k)];
                dum5=[dum5;TSang(j,k)];
                dum6=[dum6;TSvel(j,k)];
            end
        end

        all=[dum1'; dum2'; dum3'; dum4'; dum5'; dum6'];
        fprintf(fid,'%8.4f %8.4f %8.4f %8.4f %8.4f %8.4f\n',all);
        fclose(fid);
    end
end

if corrxy~=[]
  fprintf('Saving *.alg file ... \n')
  eval([ 'fid=fopen(''',pathname,'/',filename,'.alg'', ''w''); ' ]);
  fprintf(fid,'%8.4f %8.4f\n',corrxy');
  if lftcrossx~=[]
    fprintf(fid,'%8.4f %8.4f\n',lftcrossx,lftcrossy);
  end
  if vfact~=[]
    fprintf(fid,'%8.4f %8.4f\n',vfact,hfact);
  end
  if rotcentx~=[]
    fprintf(fid,'%8.4f %8.4f\n',rotcentx,rotcenty);
  end
  fclose(fid);
  clear dum
end

fprintf('Saving *.mat file ... \n')
currdir=pwd;
eval(['cd ',pathname]);
eval(['save ',filename,' pixel_width pixel_height pulse_dt min_velocity max_velocity min_angle
max_angle box_width box_height reduce rotfreq_rps xcntr_pix ycntr_pix rad_cross dist_hwire cammotion
scale Renum Rotnum zoverdh']);
eval(['cd ',currdir]);




%
%       Title:  CCspotrem.m
%       Author:  Jeffrey Bons
%       Parent program: jpbpiv3.m
%       Child subroutine(s): ginputc.m
%       Date: 4 Mar 1997
%
% This program just allows the user to remove glare spots
% from PIV images to make them easier to process.

% There must be an image available to proceed.

if (ImageWin==0)
    fprintf('No image available...\n');
else

% Select the region to remove

    figure(ImageWin);
    fprintf('Make a box around the region you wish to remove\n');
    fprintf('by mouse clicking the upper-left and lower-right\n');
    fprintf('of the region: \n');
    fprintf('Click outside the image region to finish entries.\n');

% Put in a while loop to allow multiple removals

    lim=20;
    for xx=1:lim

        [ulxrem,ulyrem]=ginput(1);
        if (ulxrem<1 | ulyrem<1 | ulxrem>nsizeimdata | ulyrem>msizeimdata)
            break
        end
        [lrxrem,lryrem]=ginput(1);
        if (lrxrem<1 | lryrem<1 | lrxrem>nsizeimdata | lryrem>msizeimdata)
            break
        end
        ulxrem=round(ulxrem);
        ulyrem=round(ulyrem);
        lrxrem=round(lrxrem);
        lryrem=round(lryrem);
        remwid=lrxrem-ulxrem+1;
        remhght=lryrem-ulyrem+1;

% Set the intensity of this region to the background value

        imdata(ulyrem:lryrem,ulxrem:lrxrem)=absmean*ones(remhght,remwid);

        fprintf('Removed spot #%d ...next?\n',xx);
    end

% Display the modified image

    set(ImageWin,'name',filename);
    clg;
    set(gca, 'position',[0 0 1 1]);
    imshow(imdata, mapimdata);

end




%
%       Title: corrimagjb2.m
%       Author:  Jeffrey Bons
%       Parent program: jpbpiv3.m
%       Child subroutine(s): descend.m, ginputc.m, corrmap.m, centroidfit.m
%       Date: 4 Mar 1997
%
% This program is used to obtained the necessary parameters
% for particle "derotation" in rotating PIV picture analysis.
% Essentially, an image with a double exposure of a rotating
% alignment tool is processed for certain pixel distances.
% The alignment tool currently used has 2 horizontal wires
% separated by a distance, dist_hwire, set in jpbpiv3 parameters
% list.  Also, between these two wires is a pair of crossed
% wires.
% The 2 horizontal wires give a distance reference in the
% vertical (y) direction, and the double exposure of the cross
% provides an excellent image for measuring the distance of
% of rotation.  corrimagjb2.m makes both of these pixel
% measurements using an image correlation.

% To use corrimagjb2.m, there must be an image already loaded into
% ImageWin using CCloadim.m.

if ImageWin==0
    fprintf('ERROR: no current ImageWin to process...load image!\n');
else
    fprintf('\nUsing ImageWin from %s...\n',filename);
    figure(ImageWin);

% Determine Image Size (image data is found in imdata from CCloadim.m)

    [mfull,nfull]=size(imdata);

% The analysis is slightly different for correlating the horiz wires
% and the cross, so define which it is...

    corrnum=0;
    while (corrnum~=1 & corrnum~=2)
        corrnum=input('Correlating cross (1) or horizontal wires (2)? ');
```

```
     end

 % Identify a finite region in the large .tif file which the pattern
 % to be identified will be correlated into.  This region must be
 % larger than the pattern region.

     fprintf('\nUse GENEROUS perimeter to define 1st region...');
     fprintf('\nClick the mouse on the upper left and lower right corners\n');
     if corrnum==1
         fprintf('of the left cross in the image\n');
     else
         fprintf('of the upper horizontal wire in the image\n');
     end

 % ginputc.m is used to input the corners of the region.  It only
 % accepts 2 mouse inputs.

     [cornerx,cornery]=ginputc(2);

 % The input needs to be upper left corner then lower right corner,
 % if it wasn't done this way, then flip the two points.

     if(cornerx(1)<cornerx(2))
         ulx=round(cornerx(1));
         uly=round(cornery(1));
         lrx=round(cornerx(2));
         lry=round(cornery(2));
     else
         ulx=round(cornerx(2));
         uly=round(cornery(2));
         lrx=round(cornerx(1));
         lry=round(cornery(1));
     end

 % Assemble the matrix of the selected region #1

     imdatareg=[];
     imdatareg=imdata(uly:lry,ulx:lrx);

     [mreg,nreg]=size(imdatareg);

 % Select pattern region within the large tif file that will be
 % correlated into region #1 identified above.

     figure(ImageWin);

     fprintf('\nUse SMALLER perimeter to define 2nd region...');
     fprintf('Click the mouse on the upper left and lower right corners\n');
     if corrnum==1
         fprintf('of the right cross in the image\n');
     else
         fprintf('of the lower horizontal wire in the image\n');
     end

     [patregx,patregy]=ginputc(2);

     if(cornerx(1)<cornerx(2))
         xstpatt=round(patregx(1));
         ystpatt=round(patregy(1));
         xendpatt=round(patregx(2));
         yendpatt=round(patregy(2));
     else
         xstpatt=round(patregx(2));
         ystpatt=round(patregy(2));
         xendpatt=round(patregx(1));
         yendpatt=round(patregy(1));
     end

     pattern=[];
     pattern=imdata(ystpatt:yendpatt,xstpatt:xendpatt);
     [rowpatt,colpatt]=size(pattern);

 % If the cross is being correlated, it must first be derotated
 % since the two cross images are rotated wrt each other.  For
 % the horizontal wire correlation, this is not necessary.
 %
 % Rotate the right cross image using a bilinear interpolation
 % method prescribed by the Image Toolbox function imrotate.m
 % Also, to avoid "1's" at the edges, clip the outer edge of
 % the pattern matrix after rotation and before doing the correlation.
```

```
% The amount of rotation is calculated from 2*pi*(rps)*deltat*(180/pi)
% This must be made negative, since imrotate rotates counterclockwise
% for a positive input angle (in degrees).  In our case, the right
% image must be rotated clockwise wrt the left image for best
% correlation.  Since pulse_dt is input in microseconds,
% must convert to seconds.

    if corrnum==1
        degrot=-360*rotfreq_rps*pulse_dt*1.0e-6;
        fprintf('Rotating right cross by %.4f degrees \n',degrot);
        rotpattern=imrotate(pattern,degrot,'bilinear','crop');
        rotpattern2=rotpattern(2:(rowpatt-1),2:(colpatt-1));

% Adjust the extent of the rotated pattern

        [rowpatt,colpatt]=size(rotpattern2);
        xstpatt=xstpatt+1;
        ystpatt=ystpatt+1;
        xendpatt=xendpatt-1;
        yendpatt=yendpatt-1;
        pattern=[];
        pattern=rotpattern2;
    else
        degrot=0;
    end

% Perform the cross-correlation of the 2 intensity maps by
% looping through imdatareg matrix with pattern, summing the
% pixel map correlations in a new matrix "D"

    D=[];
    fprintf('\nPerforming comparison...\n');
    for k=1:(mreg-rowpatt+1)
        for l=1:(nreg-colpatt+1)
            sub=imdatareg(k:k+rowpatt-1,l:l+colpatt-1);
            D(k,l)=sum(sum(sub.*pattern));
        end
    end

% The point where D has a maximum value will be the best
% image match between "imdatareg" and "pattern"

    [a,b]=max(D);
    [c,d]=max(a);
    deltax=xstpatt-(ulx+d-1);
    deltay=ystpatt-(uly+b(d)-1);

% Print out the result.

    fprintf('Rough corr gives dx=%d & dy=%d pixels\n',deltax,deltay);

% Better this with a sub-pixel correlation using an area centroid
% fit to the near-maximum portion of the D matrix.
% First find the extent of the maximum portion of the
% correlation.  descend.m steps out in 8 directions from the
% maximum point to see when the peak levels off or changes slope.

    Drat=[];
    Drat=D/max(max(D));
    dir=[0 1;-1 1;-1 0;-1 -1;0 -1;1 -1;1 0;1 1];
    steps=[];
    val=[];
    for m=1:8
        [steps(m),val(m)]=descend(Drat,d,b(d),dir(m,:));
    end

% If the max point is not on the edge, then proceed to form a region
% about this max point using corrmap.m   Corrmap.m creates a "star"-
% like map around the center point using the extents from descend.m
% and fills all other pixels in the map with background level
% intensity.  This makes for a good image to find the peak center
% via centroidfit.m

    if (min(steps)>=1)
        ind1=[8 1 2 ; 2 3 4 ; 4 5 6 ; 6 7 8];
        for l=1:4
            maxext3(l)=max([steps(ind1(l,1)) steps(ind1(l,2))...
                                            steps(ind1(l,3))]);
        end
        Dfit=[];
        [Dfit]=corrmap(Drat,d,b(d),steps,mean(val),maxext3);
```

```
% Adjust the max position (d,b(d)) in the case that size(Dfit)<
% size (Drat) to follow the true max position.

        ymaxdfit=maxext3(2)+1;
        xmaxdfit=maxext3(3)+1;

% Further refine the correlation map by zeroing out all pixels
% with a value < mean(val).  Then perform a centroid fit to the
% top 1/3rd of the peak.

        Dfit=Dfit-mean(val);
        Dfit(find(Dfit<0))=zeros(size(find(Dfit<0)));
        ff=[];
        ff=find(Dfit<(0.66*max(max(Dfit))));
        Dfit(ff)=zeros(size(ff));
        [xcntg,ycntg]=centroidfit(Dfit);

% If the max point is on the edge of the correlation region,
% then take this point as the peak and proceed.

    else
        xmaxdfit=d;
        ymaxdfit=b(d);
        xcntg=d;
        ycntg=b(d);
        fprintf('max on edge  ');
    end

    deltax2=deltax+(xmaxdfit-xcntg);
    deltay2=deltay+(ymaxdfit-ycntg);

% Print out results.  Also, corrxy is used to store the dx and dy
% distances from corrimagjb2.m for later use in derotation programs.
% (corrnum=1 for cross and corrnum=2 for horizontal wires).

    fprintf('Fine corr gives dx=%.2f & dy=%.2f pixels\n',deltax2,deltay2);
    corrxy(corrnum,1)=deltax2;
    corrxy(corrnum,2)=deltay2;

% If this is the cross being correlated, I need to know the precise
% pixel location of the left cross point in the image.  Do this by
% showing the region imdatareg in a separate window and selecting the
% center with ginputc.m

    if corrnum==1
        RegionWin=figure;
        set(RegionWin,'name','Left Cross Region');
        set(gca,'position',[0 0 1 1]);
        imshow(imdatareg,mapimdata);
        fprintf('\nSelect wire cross point in left cross image! \n');
        [inputx,inputy]=ginputc(1);
        lftcrossx=inputx-1+ulx;
        lftcrossy=inputy-1+uly;
        fprintf('Left cross at x=%.2f , y=%.2f\n',lftcrossx,lftcrossy);
        close(RegionWin);
    end

% Plot Drat if desired

    plotaway=input('Do you wish to plot Drat contour? y/n <n>:','s');
    if isempty(plotaway)
        plotaway='n';
    end

    if plotaway=='y'
        figure;
        clg
        contour(Drat);
        grid
        hold on
        plot(d,b(d),'r*');
        plot((d-xmaxdfit+xcntg),(b(d)-ymaxdfit+ycntg),'yx');
        xlabel('x or columns');
        ylabel('y or rows');
        hold off
    end

end % no ImageWin
```

```
%
%       Title:  descend.m
%       Author:  Jeffrey Bons
%       Parent program: corrpair2.m, corrimagjb2.m, CCcorrbox.m, etc...
%       Child subroutine(s):
%       Date:  28 August 1996
%
% This program searches out from the pixel (posx,posy) in
% the direction defined by dxy to see how far the function
% "data" decreases in value from the peak at (posx,posy).
% The following is returned:
% i = # of pixels to the first upturn in value in this direction
% val = function value at the "edge" in this direction

function [ i,val ] = descend(data,posx,posy,dxy)

% dy and dx can only ever have values = -1,0,+1 to define
% movement backward, none, or forward.

dy=dxy(1);
dx=dxy(2);

% Start index k, and initialize vectors

k=1;
x=[];
y=[];
v=[];
x(k)=posx;
y(k)=posy;
v(k)=data(y(k),x(k));
[mdata,ndata]=size(data);

% Move out in the direction specified, if the data border
% is breached then STOP.  Otherwise, set v(2)=intensity.

for k=2:(max([mdata ndata])+1)
    x(k)=posx+(k-1)*dx;
    y(k)=posy+(k-1)*dy;
    if ((y(k)<1) | (y(k)>mdata) | (x(k)<1) | (x(k)>ndata))
        i=k-2;
        val=v(k-1);
        return;
    end
    v(k)=data(y(k),x(k));

% If the intensity is nearly leveling off, stop moving out

    if(v(k)>=v(k-1))
        val=v(k-1);
        i=k-2;
        return;
    end
end
```

```
%
%       Title:  corrmap.m
%       Author:  Jeffrey Bons
%       Parent program: corrpair.m, CCcorrbox.m etc...
%       Child subroutine(s):
%       Date:  4 Sept 1996
%
% This program is responsible for evaluating the region
% around the pixel (posx,posy) to create
% a map of its 2D extent.

function [regpart]=corrmap(data,posx,posy,i,bckgrnd,maxext)

dir=[ 0  1 ; -1  1 ; -1  0 ; -1 -1 ; 0 -1 ; 1 -1 ; 1  0 ; 1  1];

% Create a rectangular region around posx,posy to include the
% maximum extents in each of 4 general directions contained in
% ind1: East, North, West, & South
% Fill a rectangular matrix of these dimensions with a
% background level of intensity = mean of particle edge values
```

```
regpart=[];
regpart=bckgrnd*ones((maxext(2)+maxext(4)+1),(maxext(1)+maxext(3)+1));

% Evaluate each of the 4 quadrants in the rectangle around posx,posy
% to see if the pixels in the quadrant are "within" the particle or
% "outside" of the particles 2D extent.  If within, put the intensity
% in the regpart matrix.  If without, don't do anything.
% Compute the coordinates of the last point of the particle
% extent in each of the 8 directions.

for l=1:8
    xpt(l)=posx+i(l)*dir(l,2);
    ypt(l)=posy+i(l)*dir(l,1);
end

% Compute the slopes/intercepts of the 2 defining lines (connecting
% the 3 last points) in each quadrant in data space.
% e.g. line 1 is the line connecting last point in dir 1 & 2...
% If the quadrant is convex, the line slopes can be 0 or inf
% which will cause problems later, so prepare accordingly.
% Since 0 slopes are easier to work with, lines 2,3,6,7 are
% computed with yslopes (ym) (which can be 0)
% and lines 1,4,5,8 are computed with xslopes (xm) (which can
% be 0).

for l=1:8
    if (l==8)
        k=1;
    else
        k=l+1;
    end
    if (l==2 | l==3 | l==6 | l==7)
        if ((ypt(k)-ypt(l))==0)
            ym(l)=0;
            yint(l)=ypt(l);
        else
            ym(l)=(ypt(k)-ypt(l))/(xpt(k)-xpt(l));
            yint(l)=ypt(l)-ym(l)*xpt(l);
        end
    else
        if ((xpt(k)-xpt(l))==0)
            xm(l)=0;
            xint(l)=xpt(l);
        else
            xm(l)=(xpt(k)-xpt(l))/(ypt(k)-ypt(l));
            xint(l)=xpt(l)-xm(l)*ypt(l);
        end
    end
end

% Quadrant 1: NorthEast of posx,posy (directions 1,2,& 3)

% Determine first if curve connecting 3 points is concave (points
% to center) or convex.

m13=(ypt(1)-ypt(3))/(xpt(1)-xpt(3));
int13=ypt(1)-m13*xpt(1);
if (ypt(2)<(m13*xpt(2)+int13))
    concave=0;
else
    concave=1;
end

% Now interrogate every pixel in Quadrant 1 to see if it's
% within or without the particle's extent.

for x=posx:(posx+maxext(1))
    for y=(posy-maxext(2)):posy
        if(concave==1 & ( x<=(xm(1)*y+xint(1)) | y>=(ym(2)*x+yint(2)) ) )
            regpart((maxext(2)+1+y-posy),(maxext(3)+1+x-posx))=data(y,x);
        end
        if (concave==0 & x<=(xm(1)*y+xint(1)) & y>=(ym(2)*x+yint(2)) )
            regpart((maxext(2)+1+y-posy),(maxext(3)+1+x-posx))=data(y,x);
        end
    end
end

% Quadrant 2: NorthWest of posx,posy (directions 3,4,& 5)

% Determine first if curve connecting 3 points is concave (points
% to center) or convex.
```

```
m35=(ypt(3)-ypt(5))/(xpt(3)-xpt(5));
int35=ypt(3)-m35*xpt(3);
if (ypt(4)<(m35*xpt(4)+int35))
    concave=0;
else
    concave=1;
end

% Now interrogate every pixel in Quadrant 2 to see if it's
% within or without the particle's extent.

for x=(posx-maxext(3)):posx
    for y=(posy-maxext(2)):posy
        if(concave==1 & ( y>=(ym(3)*x+yint(3)) | x>=(xm(4)*y+xint(4)) ) )
            regpart((maxext(2)+1+y-posy),(maxext(3)+1+x-posx))=data(y,x);
        end
        if (concave==0 & y>=(ym(3)*x+yint(3)) & x>=(xm(4)*y+xint(4)) )
            regpart((maxext(2)+1+y-posy),(maxext(3)+1+x-posx))=data(y,x);
        end
    end
end

% Quadrant 3: SouthWest of posx,posy (directions 5,6,& 7)

% Determine first if curve connecting 3 points is concave (points
% to center) or convex.

m57=(ypt(7)-ypt(5))/(xpt(7)-xpt(5));
int57=ypt(5)-m57*xpt(5);
if (ypt(6)>(m57*xpt(6)+int57))
    concave=0;
else
    concave=1;
end

% Now interrogate every pixel in Quadrant 3 to see if it's
% within or without the particle's extent.

for x=(posx-maxext(3)):posx
    for y=posy:(posy+maxext(4))
        if(concave==1 & ( y<=(ym(6)*x+yint(6)) | x>=(xm(5)*y+xint(5)) ) )
            regpart((maxext(2)+1+y-posy),(maxext(3)+1+x-posx))=data(y,x);
        end
        if (concave==0 & y<=(ym(6)*x+yint(6)) & x>=(xm(5)*y+xint(5)) )
            regpart((maxext(2)+1+y-posy),(maxext(3)+1+x-posx))=data(y,x);
        end
    end
end

% Quadrant 4: SouthEast of posx,posy (directions 7,8,& 1)

% Determine first if curve connecting 3 points is concave (points
% to center) or convex.

m71=(ypt(7)-ypt(1))/(xpt(7)-xpt(1));
int71=ypt(7)-m71*xpt(7);
if (ypt(8)>(m71*xpt(8)+int71))
    concave=0;
else
    concave=1;
end

% Now interrogate every pixel in Quadrant 4 to see if it's
% within or without the particle's extent.

for x=posx:(posx+maxext(1))
    for y=posy:(posy+maxext(4))
        if(concave==1 & ( y<=(ym(7)*x+yint(7)) | x<=(xm(8)*y+xint(8)) ) )
            regpart((maxext(2)+1+y-posy),(maxext(3)+1+x-posx))=data(y,x);
        end
        if (concave==0 & y<=(ym(7)*x+yint(7)) & x<=(xm(8)*y+xint(8)) )
            regpart((maxext(2)+1+y-posy),(maxext(3)+1+x-posx))=data(y,x);
        end
    end
end



%
```

```
%       Title: centroidfit.m
%       Author:  Jeffrey Bons
%       Parent program: corrpair4.m, corrimagjb2.m, CCcorrbox.m, etc...
%       Child subroutine(s): ycentroid.m & xcentroid.m
%       Date:  18 July 1996
%
% centroidfit.m is the next step in the fitting process.  It basically
% puts the data into a form which can be easily fitted with a
% intensity centroid using the Matlab minimization function fmins
% and ycentroid.m & xcentroid.m
% centroidfit.m returns all the relevant gaussian data to ptypejpb.m
% which simply passes it back to getpartjb.m

function [xct,yct] =centroidfit(centdata)

% centdata contains the grayscale pixel data in the region
% surrounding the particle being evaluated.  First determine
% its size and then reshape the data into a column vector.
% (e.g. for 3x3 might have [57 85 60 80 180 88 70 90 40])

[szm,szn]=size(centdata);
szcentdata=szm*szn;
z=(reshape(centdata,1,szcentdata))';

% Create a similar column vector of x's, essentially giving the
% pixel distance from the leftmost column of the region centdata.
% (e.g. for 3x3 "x" would be [0 0 0 1 1 1 2 2 2])

x=zeros(szm,szn);
   for i=2:szn,
     x(:,i)=(i-1)*ones(szm,1);
   end
x = (reshape(x,1,szcentdata))';

% Create a column vector of y's, essentially giving the
% pixel distance from the topmost row of the region centdata.
% (e.g. for 3x3 "y" would be [0 1 2 0 1 2 0 1 2])

y=zeros(szm,szn);
   for i=2:szm,
     y(i,:)=(i-1)*ones(1,szn);
   end
y = (reshape(y,1,szcentdata))';

% NOTE:  All of this reshaping into vectors is not necessary
% for running the minimization routine which can work just
% fine on matrices.  However, using vectors speeds the process
% up by about 100%!!!

% fmin is defined in the Optimization Toolbox of Matlab.  It
% is used to find the minimum of a scalar function of one
% variable.

global x y z

[yct,out] = fmin('ycentroid',0,szm-1);
yct=yct+1;

[xct,out] = fmin('xcentroid',0,szn-1);
xct=xct+1;

clear global x y z
```

```
%
%       Title: xcentroid.m
%       Author:  Jeffrey Bons
%       Parent program: centroidfit.m
%       Child subroutine(s):
%       Date:  18 July 1996
%
% This very short function is the one minimized in fmin
% of centroidfit.m

function err = xcentroid(xoffset)

global x z

err = abs(sum(z.*(x-xoffset)));
```

```
%
%       Title: ycentroid.m
%       Author:  Jeffrey Bons
%       Parent program: centroidfit.m
%       Child subroutine(s):
%       Date:  18 July 1996
%
% This very short function is the one minimized in fmin
% of centroidfit.m

function err = ycentroid(yoffset)

global y z

err = abs(sum(z.*(y-yoffset)));
```

```
%
%       Title: xyresol.m
%       Author:  Jeffrey Bons
%       Parent program: jpbpiv3.m
%       Child subroutine(s):
%       Date: 5 Mar 1997
%
%  Since the CCD camera we use, TM745E, does not have square pixels
%  I need to determine the pixel resolution in both the vertical
%  and horizontal directions.  This routine computes these two
%  resolutions based on measurements made in real space, and
%  comparing these to distances in the pixel map.

% Several parameters are needed to perform this function

if (rotfreq_rps==[] | rad_cross==[] | pulse_dt==[] |...
    dist_hwire==[] | corrxy==[])
    fprintf('ERROR: Set parameters first: rps,deltat,disthwire,rad\n');
else

% "corrxy" contains distances measured by corrimagjb2.m for
% the wire cross double exposure and the horizontal alignment
% wires respectively:

    delxcr=corrxy(1,1);
    delycr=corrxy(1,2);
    delxhor=corrxy(2,1);
    delyhor=corrxy(2,2);

% First, the cross in the alignment tool is at a radius that
% has been measured  in microns (see notebook, 27 Jun &
% 1 & 9 July 96) and is set in jpbpiv3 as rad_cross.
% Since the cross image is a double exposure, and I know the
% angular velocity and the time between exposures, I can
% compute the linear distance between the 2 crosses as
% 2*rad*sin{(rps) * deltat/2}:
% rotfreq_rps and pulse_dt are parameters in jpbpiv3
% pulse_dt must be converted from microseconds to sec

    sec1=2*rad_cross*sin(2*pi*rotfreq_rps*pulse_dt*1.0e-6/2);

% The distance between the 2 horizontal alignment wires is
% measured in microns and is set in jpbpiv1 as dist_hwire.

% Now I simply solve the 2 quadratics to get the horizontal
% and vertical pixel resolutions in micrometers/pixel (see
% notebook 10 Jul 96):

    num=delxcr^2*dist_hwire^2-delxhor^2*sec1^2;
    denom=delxcr^2*delyhor^2-delxhor^2*delycr^2;
    vfact=sqrt(num/denom);

    hfact=sqrt((sec1^2-delycr^2*vfact^2)/delxcr^2);

% And the ratio is:

    ratfact=vfact/hfact;
    fprintf('Pixel dimensions: vertical = %.4f microns \n',vfact);
    fprintf('\t\thorizontal = %.4f microns \n',hfact);
    fprintf('\t\tvertical/horizontal ratio = %.4f \n',ratfact);
end



%
%       Title: rpsdet.m
%       Author: Jeffrey Bons
%       Parent program:
%       Child subroutine(s):
%       Date: 25 Mar 1997
%
% This program determines the rps based on the TS image
% position in the reference frame.

% frame pixel dimensions are 752 x 480

% reference conditions

% for 21 Mar data
rpsedge=9.9995;
rpscntr=10.0175;
delrps=2.57e-5;

% for 3 Mar data
%rpscntr=10;
%delrps=.0125;

side=input('Which side of TS is identified? l or r <r>:','s');

figure(ImageWin);

fprintf('click on specified side\n');
[xedge,yedge]=ginput(1)

if side=='l'
        rpsout=rpsedge+delrps*(xedge+650);
%       rpsout=rpscntr+delrps*(xedge-60)/(376-60); % 3 Mar
else
        rpsout=rpsedge+delrps*xedge;
%       rpsout=rpscntr-delrps*(692-xedge)/(692-376); % 3 Mar
end

% Print the results to screen

fprintf('\nResult is: \trps=%f',rpsout);

% If desired, input these into the jpbpiv3 graphical
% user interface.

ansa=input('Input new data into jpbpiv3? y/n <y>: ','s');
if isempty(ansa)
    ansa='y';
end

if ansa=='y'
    rotfreq_rps=rpsout;
    set(ImAttrib_edit41,'string',num2str(rotfreq_rps));
    fprintf('New parameter input to jpbpiv3..\n');
end



%
%       Title: pixmag.m
%       Author:  Jeffrey Bons
%       Parent program: jpbpiv3.m
%       Child subroutine(s):
%       Date: 25 Mar 1997
%
%  Anytime I adjust the camera to focus on the current light
% sheet position, the magnification changes from the alignment
% magnification calculated.  This routine computes the needed
% adjustment and the change in pixel center of rotation.

% The first thing to do is verify the alignment pixel dimensions
% and center of rotation, as well as the location of alignment.
% Data from 28 Feb 97 adjusted for 21 Mar movement of optics
Datadate='28 Feb 97';
algpixht=16.80;  %microns
algpixwd=14.29;  %microns
algxcntr=525;     %pixels
algycntr=-25893;  %pixels
algoptpos=0.845;    %inches 21 Mar data
%algoptpos=1.0256;    %inches 3 Mar data
algcampos=0.143;    %inches

fprintf('\nThe alignment results currently in use are');
fprintf('\nfrom %s :',Datadate);
fprintf('\npixel height (h) = %f',algpixht);
fprintf('\npixel width (w) = %f',algpixwd);
fprintf('\npixel cntrofrot (xc) = %f',algxcntr);
fprintf('\npixel cntrofrot (yc) = %f',algycntr);

% Change in pixel dimension with sheet motion was measured on
% 28 Feb 97 to be 0.82 percent per mm of motion.

magadj=0.82;  % percent/mm

% When moving the camera traverse, the xc,yc pixels move also.
% I've measured this motion a few times, the yc motion is pretty
% repeatable, while the xc is not so...
% Using the data from 28 Feb 97

ycntradj=-1515;  %ypixels/inch camera motion
xcntradj=-54;    %xpixels/inch camera motion

% Also need an estimate of test section center for z/d calc.
% Data from 28 Feb 97 measurement (adjusted for 21 March)

tscntr=[0.846 0.846];  %inches on optics traverse on 21 Mar data
%tscntr=[1.026 1.026];  %inches on optics traverse on 3 Mar data
camposdat=[0.143 1.8];     %inches on camera traverse corresponding to tccntr

% Now that all of that's input, the computation is pretty
% straightforward.

% First input the optics traverse position for this data...

optloc=input('Enter optics traverse position (inches): ');

% Compute new pixel dimensions

Magnif=1+(optloc-algoptpos)*25.4*magadj/100;
actpixht=algpixht*Magnif;
actpixwd=algpixwd*Magnif;

% Next input the camera traverse position for this data...

camloc=input('Enter camera traverse position (inches): ');

% Compute z/d position

acttscntr=((tscntr(2)-tscntr(1))/(camposdat(2)-camposdat(1)))...
                *(camloc-camposdat(1))+tscntr(1);

actzoverd=(optloc-acttscntr)*25.4/10;

% Compute actual xc,yc

xcint=xcntradj*(camloc-algcampos)+algxcntr;
ycint=ycntradj*(camloc-algcampos)+algycntr;

% To adjust the xc,yc for the magnification, the pixel distance
% from the center of the optical axis to the current xc,yc
% must be adjusted.

actxc=(xcint-376)/Magnif+376;
actyc=(ycint-240)/Magnif+240;

% Compute camera motion from alignment position in microns

cammotion=(camloc-algcampos)*25.4*1000;

% Print the results to screen

fprintf('\nResults are: \tz/d=%f',actzoverd);
fprintf('\n\t\tpixel h=%f',actpixht);
fprintf('\n\t\tpixel w=%f',actpixwd);
fprintf('\n\t\txc=%f',actxc);
fprintf('\n\t\tyc=%f',actyc);
fprintf('\n\t\tcammotion=%f\n',cammotion);

% If desired, input these into the jpbpiv3 graphical
% user interface.

ansa=input('Input new data into jpbpiv3? y/n <y>: ','s');
if isempty(ansa)
    ansa='y';
end

if ansa=='y'
    pixel_width=actpixwd;
    pixel_height=actpixht;
    ycntr_pix=actyc;
    xcntr_pix=actxc;
    zoverdh=actzoverd;
    set(ImAttrib_edit14,'string',num2str(pixel_width));
    set(ImAttrib_edit15,'string',num2str(pixel_height));
    set(ImAttrib_edit42,'string',num2str(xcntr_pix));
    set(ImAttrib_edit43,'string',num2str(ycntr_pix));
    set(ImAttrib_edit46,'string',num2str(cammotion));
    set(ImAttrib_edit62,'string',num2str(zoverdh));
    fprintf('New parameters input to jpbpiv3..\n');
end




%
%       Title: CCbox.m
%       Author: Jeffrey Bons
%       Parent program: jpbpiv3.m
%       Child subroutine(s): ginputc.m
%       Date: 4 Mar 1997
%
% This program divides a user defined rectangle in the
% ImageWin into small boxes for cross-correlation.
%
% Can't run without parameters for box dimensions

if( box_width==[] | box_height==[] )
    fprintf('ERROR: set box dimensions first!!\n');
else

% Bring up the ImageWin and print the current rectangle
% corner values (if any).  User can select the corners of a
% rectangle encompassing the left image, enter them from
% the keyboard, or accept the current values.

    if (ImageWin==0)
        fprintf('ERROR: no ImageWin available...\n\n');
    else
        figure(ImageWin);

        if (ulxrect~=[] & ulyrect~=[] & lrxrect~=[] & lryrect~=[])
            fprintf('Current rectangle coordinates are: \n');
            fprintf(' upper left: x=%d  y=%d \n',ulxrect,ulyrect);
            fprintf('lower right: x=%d  y=%d \n',lrxrect,lryrect);
        else
            fprintf('No rectangle currently specified!\n');
        end

        fprintf('\nChoose from selections for inputing rectangle:');
        ansc=input('c=use current, m=mouse input, k=keyboard input <c>:','s');
        if isempty(ansc)
            ansc='c';
        end

        if ansc=='m'
            fprintf('Create rectangle surrounding left image\n');
            fprintf('Select upper left corner of rectangle.\n');
            [ulxrect,ulyrect]=ginputc(1);
            ulxrect=round(ulxrect);
            ulyrect=round(ulyrect);
            fprintf('Select lower right corner of rectangle.\n');
            [lrxrect,lryrect]=ginputc(1);
            lrxrect=round(lrxrect);
            lryrect=round(lryrect);
        elseif ansc=='k'
            ulxrecto=ulxrect;
            ulyrecto=ulyrect;
            lrxrecto=lrxrect;
            lryrecto=lryrect;
            ulxrect=input('Enter upper left x coord <current> : ');
            if isempty(ulxrect) ulxrect=ulxrecto; end;
            ulyrect=input('Enter upper left y coord <current> : ');
            if isempty(ulyrect) ulyrect=ulyrecto; end;
            lrxrect=input('Enter lower right x coord <current> : ');
            if isempty(lrxrect) lrxrect=lrxrecto; end;
            lryrect=input('Enter lower right y coord <current> : ');
            if isempty(lryrect) lryrect=lryrecto; end;
        else
            fprintf('Using current values...\n');
        end

% Limit region to available image...

        if (ulyrect<0) ulyrect=0; end;
        if (lryrect>msizeimdata) lryrect=msizeimdata; end;
        if (ulxrect<0) ulxrect=0; end;
        if (lrxrect>nsizeimdata) lrxrect=nsizeimdata; end;

        if (lrxrect < ulxrect | lryrect < ulyrect)
            fprintf('Rectangle not properly dimensioned\n');
        else

% Compute number of boxes region can hold

            nbox=floor((lrxrect-ulxrect+1)/box_width);
            mbox=floor((lryrect-ulyrect+1)/box_height);
            fprintf('Region is %d boxes wide & %d boxes tall\n',nbox,mbox);

% Subtract off the background illumination level in this section of
% the image, and stretch the current data to fit the 8-bit window

            avgboxint=mean(mean(imdata(ulyrect:lryrect,ulxrect:lrxrect)));
            fprintf('Average box intensity here is %f\n',avgboxint);
            ansm='n';
%           ansm=input('Stretch data? y/n <n>: ','s');
            if isempty(ansm)
                ansm='n';
            end

            if ansm=='y'
                imdata=round(256*(imdata-avgboxint)/(256-avgboxint));
                imdata(find(imdata<7))=7*ones(size(find(imdata<7)));
            end

% Draw boxes on selected portion of ImageWin

            if (LTSWin==0)
                LTSWin=figure;
            else figure(LTSWin);
            end

            set(LTSWin,'name',filename);
            clg;
            set(gca, 'position',[0 0 1 1]);
            imshow(imdata(ulyrect:lryrect,ulxrect:lrxrect),mapimdata);
            set(gca, 'xtick',([0:nbox]*box_width+0.5));
            set(gca, 'ytick',([0:mbox]*box_height+0.5));
            grid on

% Create xbox & ybox vectors containing the pixel coordinates
% of box centers.  For these center coordinates to be integers
% the box_width and box_height MUST be odd integers.

            xbox=[];
            ybox=[];
            xbox(1)=ulxrect+0.5*(box_width-1);
            ybox(1)=ulyrect+0.5*(box_height-1);
            for k=2:nbox
                xbox(k)=xbox(k-1)+box_width;
            end
            for k=2:mbox
                ybox(k)=ybox(k-1)+box_height;
            end
        end
```

```
% List options for optimum grid refinement

        fprintf('Possible configurations for different box #s: \n');
        fprintf('# box\tpixels wide\tpixels high\n');

        for g=(min(nbox,mbox)-2):(max(nbox,mbox)+2)
            widmin=floor((lrxrect-ulxrect+1)/g);
            hgtmin=floor((lryrect-ulyrect+1)/g);
            fprintf('%d\t%d\t\t%d\n',g,widmin,hgtmin);
        end

    end
end




%
%       Title: CCcorrbox3.m
%       Author:  Jeffrey Bons
%       Parent program: jpbpiv3.m
%       Child subroutine(s): centroidfit.m, descend.m, jpbquiv.m,
%                            rotpart.m, &corrmap.m
%       Date: 25 Mar 1997
%
% This program is used to correlate boxes from the left image
% into the right image.  The peak then indicates the distance
% and direction of motion.

% Must have box coordinates and parameters to proceed.

if (xbox==[] | pixel_width==[] | pixel_height==[] | box_width==[] |...
    box_height==[] | pulse_dt==[] | max_velocity==[] |...
    min_velocity==[] | min_angle==[] | max_angle==[] |...
    rotfreq_rps==[] | xcntr_pix==[] | ycntr_pix==[] | absmean==[] |...
    ulxrect==[] | ulyrect==[])
    fprintf('Error: Need boxes and parameters first!!\n');
else

% Determine the angular motion in radians, theta=2*pi*rps*deltat
% Convert pulse_separation from microseconds to seconds

    theta=2*pi*rotfreq_rps*pulse_dt*1.0e-6;

% Loop through all of the boxes to do correlation.

    Ddata=[];
    Pksdata=[];
    [mmax,nmax]=size(imdata);
    difx=[];
    dify=[];
    ang=[];
    vel=[];
    goodbox=zeros(length(ybox),length(xbox));
    lftxnew=[];
    lftynew=[];
    for k=1:length(xbox)
        fprintf('Column: x=%d\n',k);
        for j=1:length(ybox)
            lftxold=xbox(k);
            lftyold=ybox(j);

% Rotate the coordinates of the center of the box into the
% right test section image by angle theta.  If there's no rotation
% then new=old.

            if (abs(theta)<=0.001)
                lftxnew(j,k)=lftxold;
                lftynew(j,k)=lftyold;
            else
[lftxnew(j,k),lftynew(j,k)]=rotpart(lftxold,lftyold,theta,xcntr_pix,ycntr_pix,pixel_width,pixel_heig
ht);
            end

% The left box region will be correlated into an area in the
% right image defined by min/max velocity & angle.  Both
% regions contain image data - the mean intensity.

            boxreg=[];
            boxreg=imdata((lftyold-0.5*(box_height-1)): ...
               (lftyold+0.5*(box_height-1)),(lftxold-0.5*(box_width-1)): ...
               (lftxold+0.5*(box_width-1)))-absmean;

            if min_velocity>=0.0
              yst=round(lftynew(j,k)+min_velocity*pulse_dt...
               *cos(max([abs(90-max_angle) abs(90-min_angle)])...
                *pi/180)/pixel_height-0.5*(box_height-1));
            else
              yst=round(lftynew(j,k)+min_velocity*pulse_dt/pixel_height...
                       -0.5*(box_height-1));
            end

% If the limits don't include 90 degrees, need to adjust the
% region.

            if (min_angle<=90 & max_angle>=90)
                yend=round(lftynew(j,k)+max_velocity...
                    *pulse_dt/pixel_height+0.5*(box_height-1));
                xst=round(lftxnew(j,k)-max_velocity*pulse_dt...
                    *sin((max_angle-90)*pi/180)/pixel_width-0.5*(box_width-1));
                if min_angle>=0
                  xend=round(lftxnew(j,k)+max_velocity*pulse_dt...
                    *sin((90-min_angle)*pi/180)/pixel_width+0.5*(box_width-1));
                else
                  xend=round(lftxnew(j,k)+max_velocity*pulse_dt...
                    /pixel_width+0.5*(box_width-1));
                end
            else
                yend=round(lftynew(j,k)+max_velocity*pulse_dt...
                    *cos(min([abs(90-max_angle) abs(90-min_angle)])...
                    *pi/180)/pixel_height+0.5*(box_height-1));
                if(min_angle>90)
                    xst=round(lftxnew(j,k)-max_velocity*pulse_dt...
                        *sin((max_angle-90)*pi/180)/pixel_width...
                        -0.5*(box_width-1));
                    xend=round(lftxnew(j,k)-min_velocity*pulse_dt...
                        *sin((min_angle-90)*pi/180)/pixel_width...
                                +0.5*(box_width-1));
                else
                    xst=round(lftxnew(j,k)+min_velocity*pulse_dt...
                        *sin((90-max_angle)*pi/180)/pixel_width...
                        -0.5*(box_width-1));
                    xend=round(lftxnew(j,k)+max_velocity*pulse_dt...
                        *sin((90-min_angle)*pi/180)/pixel_width...
                        +0.5*(box_width-1));
                end
            end

% Send these 4 coordinates to the Ddata matrix for use in the
% recorrelation later.

            w=(k-1)*length(ybox)+j;
            Ddata(w,1:6)=[yst yend xst xend 0 0];

% If this region extends beyond the limits of imdata, then can't
%  process...

            if (xend>nmax | xst<1 | yst<1 | yend>mmax)
                difx(j,k)=0.0;
                dify(j,k)=0.0;
                ang(j,k)=0.0;
                vel(j,k)=0.0;
            else
                mapreg=[];
                mapreg=imdata(yst:yend,xst:xend)-absmean;

% Correlate boxreg into mapreg to determine the cross-correlation
% map D.

                D=[];
                [mmap,nmap]=size(mapreg);
                for h=1:(mmap-box_height+1)
                    for l=1:(nmap-box_width+1)
                        match=[];
                        match=mapreg(h:(h+box_height-1),l:(l+box_width-1));
                        D(h,l)=sum(sum(match.*boxreg));
                    end
                end

% Make the minimum D value = 0
```

```
               D=D-min(min(D));

% Store the D matrix in the Ddata archive for later use

               [Dm,Dn]=size(D);
               Ddata(w,5:6)=[Dm Dn];
               Ddata(w,7:(6+Dm*Dn))=reshape(D,1,Dm*Dn);

% Look for peaks in the D map before finding the maximum peak

               if (min(size(D))<3)
                   [a,b]=max(D);
                   [c,d]=max(a);
                   difx(j,k)=xst+d-1+0.5*(box_width-1)-lftxnew(j,k);
                   dify(j,k)=yst+b(d)-1+0.5*(box_height-1)-lftynew(j,k);
                   compl=difx(j,k)*pixel_width+dify(j,k)*pixel_height*i;
                   ang(j,k)=angle(compl)*180/pi;
                   vel(j,k)=abs(compl)/pulse_dt;
                   goodbox(j,k)=1;
               else
                   corrpeaks=zeros(size(D));
                   for p=2:(Dm-1)
                       for q=2:(Dn-1)
                           if(D(p,q)>=D(p+1,q+1) & D(p,q)>=D(p,q+1) &...
                              D(p,q)>=D(p-1,q+1) & D(p,q)>=D(p+1,q) &...
                              D(p,q)>=D(p+1,q-1) & D(p,q)>=D(p,q-1) &...
                              D(p,q)>=D(p-1,q-1) & D(p,q)>=D(p-1,q) )
                               corrpeaks(p,q)=1;
                           end
                       end
                   end

% Store the corrpeaks matrix in the Pksdata archive for later use

               Pksdata(w,1:(Dm*Dn))=reshape(corrpeaks,1,Dm*Dn);

% If there were no peaks, then there's no vector

                   if find(corrpeaks)==[]
                       difx(j,k)=0.0;
                       dify(j,k)=0.0;
                       ang(j,k)=0.0;
                       vel(j,k)=0.0;
                   else

% Find the maximum value of D.  All entries that weren't
% determined to be peaks are not considered.

                       [a,b]=max(D.*corrpeaks);
                       [c,d]=max(a);

% Better this with a sub-pixel correlation using an area centroid
% fit to the near-maximum portion of the D matrix.
% First find the extent of the maximum portion of the
% correlation using descend.m

                       Drat=[];
                       Drat=D/max(max(D));
                       dir=[0 1;-1 1;-1 0;-1 -1;0 -1;1 -1;1 0;1 1];
                       steps=[];
                       val=[];
                       for m=1:8
                         [steps(m),val(m)]=descend(Drat,d,b(d),dir(m,:));
                       end

% Proceed to form a region about this max point using corrmap.m

                       ind1=[8 1 2 ; 2 3 4 ; 4 5 6 ; 6 7 8];
                       for l=1:4
                           maxext3(l)=max([steps(ind1(l,1))...
                                steps(ind1(l,2))  steps(ind1(l,3))]);
                       end
                       Dfit=[];
                       [Dfit]=corrmap(Drat,d,b(d),steps,mean(val),maxext3);

% Adjust the max position (d,b(d)) in the case that size(Dfit)<
% size (Drat) to follow the true max position.

                       ymaxdfit=maxext3(2)+1;
                       xmaxdfit=maxext3(3)+1;
```

```
% Further refine the correlation map by zeroing out all pixels
% with a value < mean(val).  Then perform a centroid fit to only
% the top 25% of the peak using centroidfit.m

                        Dfit=Dfit-mean(val);
                        Dfit(find(Dfit<0))=zeros(size(find(Dfit<0)));
                        ff=[];
                        ff=find(Dfit<(0.75*max(max(Dfit))));
                        Dfit(ff)=zeros(size(ff));
                        [xcntg,ycntg]=centroidfit(Dfit);

% Store the resulting correlation refinement in difx,dify

                        difx(j,k)=xst+d-1+xcntg-xmaxdfit+0.5*(box_width-1)-lftxnew(j,k);
                        dify(j,k)=yst+b(d)-1+ycntg-ymaxdfit+0.5*(box_height-1)-lftynew(j,k);
                        compl=difx(j,k)*pixel_width+dify(j,k)*pixel_height*i;
                        ang(j,k)=angle(compl)*180/pi;
                        vel(j,k)=abs(compl)/pulse_dt;
                        goodbox(j,k)=1;
                        if(vel(j,k)>max_velocity | vel(j,k)<min_velocity |...
                            ang(j,k)>max_angle | ang(j,k)<min_angle)
                            difx(j,k)=0.0;
                            dify(j,k)=0.0;
                            ang(j,k)=0.0;
                            vel(j,k)=0.0;
                            goodbox(j,k)=0;
                        end
                    end
                end
            end
        end
    end

% Determine the total number of boxes that were zero'd

    fprintf('Total boxes with zeros = %d out of
%d\n',length(find(goodbox==0)),length(xbox)*length(ybox));
    fprintf('or %.1f percent\n',100*length(find(goodbox==0))/(length(xbox)*length(ybox)));

    if (length(find(goodbox==0))==length(xbox)*length(ybox))
        fprintf('No vectors found...\n');
        goodbox=[];
        break;
    end

% Show on LTSWin plot the boxes that have data and those
% that have "zeros".

    if LTSWin==0
        fprintf('No LTSWin available for displaying boxes \n');
    else
        figure(LTSWin);
        clg
        set(gca, 'position', [0 0 1 1]);
        imshow(imdata(ulyrect:lryrect,ulxrect:lrxrect),mapimdata);
        mbox=length(ybox);
        nbox=length(xbox);
        set(gca, 'xtick',([0:nbox]*box_width+0.5));
        set(gca, 'ytick',([0:mbox]*box_height+0.5));
        set(gca, 'yticklabels',[1:mbox]);
        grid on
        hold on
        xzero=[];
        yzero=[];
        for k=1:length(xbox)
            for j=1:length(ybox)
                if goodbox(j,k)==0
                   xzero=[xzero;(xbox(k)-ulxrect+1)];
                   yzero=[yzero;(ybox(j)-ulyrect+1)];
                end
            end
        end

% These vectors are shown in color cyan

        jpbquiv((xbox-ulxrect+1),(ybox-ulyrect+1),difx,dify,scale,'-c');
        if xzero~=[]
            plot(xzero,yzero,'ro');
        end
        hold off
```

```
     end

 % Open a HistWin for plotting velocity & angle histograms

     if (HistWin==0)
         HistWin=figure;
         orient landscape
     else
         figure(HistWin);
         clf
     end
     set(HistWin,'name',filename);

 % Plot angle histogram

     axes('units','normalized','position',[.05 .1 .9 .3])
     hist(ang(find(goodbox)),30);
     set(gca, 'xtick',[-180:10:180]);
     grid on
     set(gca, 'xlim',[(min(min(ang(find(goodbox))))-5) (max(max(ang(find(goodbox))))+5)]);
     mang=num2str(mean(mean(ang(find(ang)))));
     nmin=num2str(min_angle);
     nmax=num2str(max_angle);
     title(['mean=',mang,' min=',nmin,' max=',nmax]);
     xlabel('Angles in degrees');

 % Plot velocity histogram

     axes('units','normalized','position',[.05 .55 .9 .3])
     hist(vel(find(goodbox)),30);
     set(gca, 'xtick',[-5:.5:10]);
     grid on
     set(gca, 'xlim',[min_velocity max_velocity]);
     mvel=num2str(mean(mean(vel(find(vel)))));
     nmin=num2str(min_velocity);
     nmax=num2str(max_velocity);
     title(['mean=',mvel,' min=',nmin,' max=',nmax]);
     xlabel('Velocity in m/s');

% Pause for a second so that the screen has a chance to plot the
% HistWin before moving on...

    pause(1);

end



%
%       Title:  jpbquiv.m
%       Author:  Jeffrey Bons
%       Parent program: CCcorrbox2.m and others
%       Child subroutine(s): xyzchk.m & lower.m
%       Date: 4 Mar 1997
%
% This program is used to plot the vectors.  It is a
% modified version of Matlab's original quiver.m.  The only
% difference is that this version has a fixed scaling vs.
% the autoscaling provided in quiver.m

% The input arguments are as follows:
% arg1 = x position of vector
% arg2 = y position of vector
% arg3 = dx of vector
% arg4 = dy of vector
% arg5 = scaling factor
% arg6 = line style for plotting (e.g. '-y'=  yellow sold line)

function hh = jpbquiv(arg1,arg2,arg3,arg4,arg5,arg6)
error(nargchk(2,6,nargin));

% Arrow head parameters
alpha = 0.33; % Size of arrow head relative to the length of the vector
beta = 0.33;  % Width of the base of the arrow head relative to the length

% xyzchk.m does x,y,u,v calculation

[msg,x,y,u,v] = xyzchk(arg1,arg2,arg3,arg4);

if ~isempty(msg), error(msg); end
if nargin~=6, % jpbquiv(x,y,u,v,scale,style)
    error('need 6 inputs');
else

    u = u*arg5;
    v = v*arg5;

    ax = newplot;
    next = lower(get(ax,'NextPlot'));
    hold_state = ishold;

% Make velocity vectors

    x = x(:).'; y = y(:).';
    u = u(:).'; v = v(:).';
    uu = [x;x+u;NaN*ones(size(u))];
    vv = [y;y+v;NaN*ones(size(u))];

    sym = arg6;
    h = plot(uu(:),vv(:),sym);

% Make arrow heads and plot them

    hu = [x+u-alpha*(u+beta*(v+eps));x+u; ...
      x+u-alpha*(u-beta*(v+eps));NaN*ones(size(u))];
    hv = [y+v-alpha*(v-beta*(u+eps));y+v; ...
      y+v-alpha*(v+beta*(u+eps));NaN*ones(size(v))];
    hold on
    h = [h;plot(hu(:),hv(:),sym)];

    if ~hold_state, hold off, view(2); set(ax,'NextPlot',next); end

    if nargout>0, hh = h; end
end
```

```
%
%       Title: rotpart.m
%       Author: Jeffrey Bons
%       Parent program: rotplist.m & CCcorrbox2.m
%       Child subroutine(s):
%       Date: 22 July 1996
%
% This program solves for the new location of a point
% (xnewpix,ynewpix) given its old coordinates (xoldpix,yoldpix), the
% angular motion (theta in radians) and the center of
% rotation coordinates (xcpix,ycpix).  The pixel dimensions are
% needed to perform the rotation.

function [xnewpix,ynewpix]=rotpart(xoldpix,yoldpix,theta,xcpix,ycpix,pixwid,pixht)

% Transfer all lengths from pixels to micrometers

xold=xoldpix*pixwid;
xc=xcpix*pixwid;
yold=yoldpix*pixht;
yc=ycpix*pixht;

% Compute the radius of the point (distance from xc,yc)

rad=sqrt((xold-xc)^2+(yold-yc)^2);

% Need to iterate to find the new coordinates.
% This is set up for measuring positive theta motion from
% the upper left of the figure to the bottom right of the
% figure.  So, generally it is assumed that xnew>xold and
% ynew>yold.  Solve the equations accordingly...

xnew1=xold+50;
del=1;
it=0;
while (del>1e-9 & it<500)
    ynew=yc+sqrt(rad^2-(xnew1-xc)^2);
    xnew2=xold+sqrt((2*rad*sin(theta/2))^2-(ynew-yold)^2);
    del=abs((xnew1-xnew2)/xnew2);
    xnew1=xnew2;
    it=it+1;
end
```

```
% With this value for xnew, solve for ynew and convert
% back to pixels

xnew=xnew1;
ynew=yc+sqrt(rad^2-(xnew-xc)^2);

xnewpix=xnew/pixwid;
ynewpix=ynew/pixht;

clear xnew ynew xnew1 xnew2 rad xc yc yold xold




%
%       Title: CCrecorrbox4.m
%       Author:  Jeffrey Bons
%       Parent program: jpbpiv3.m
%       Child subroutine(s): centroidfit.m, descend.m, & corrmap.m
%       Date: 25 Mar 1997
%
% This program is used to have a second look at boxes which
% didn't correlate well in CCcorrbox3.m.

% Must have vel & ang data & box coordinates & parameters to proceed.

if (xbox==[] | pixel_width==[] | pixel_height==[] | box_width==[] |...
    box_height==[] | pulse_dt==[] | max_velocity==[] |...
    min_velocity==[] | min_angle==[] | max_angle==[] |...
    rotfreq_rps==[] | xcntr_pix==[] | ycntr_pix==[] | absmean==[] | ...
    ang==[] | vel==[] | lftxnew==[] | lftynew==[] | reduce==[] |...
    goodbox==[] | ulyrect==[] | ulxrect==[] | lryrect==[] | lryrect==[])
    fprintf('Error: Need velocities, boxes and parameters first!!\n');
else

% Print the status from CCcorrbox3 before recorrelating

  fprintf('\nStep\tOld->New\tZeros\tBoxes\tPercent');
  numz=length(find(ang==0 & vel==0));

fprintf('\n1.0\t0\t\t%d\t%d\t%.0f',numz,length(xbox)*length(ybox),(100*numz/length(xbox)/length(ybox
)));

% Loop through this program, each time lowering "zzz" so that
% the search region is being narrowed each time in the hope of
% finding a good vector (or one that is closer to its neighbors
% than the one found in CCcorrbox3.

  redo=ones(size(goodbox));
  changed=zeros(size(goodbox));
  numnewtot=0;
  for zzz=0.9:-0.1:0.3
    numnew=0;

% Here's the for loop for boxes, k & j

    [mmax,nmax]=size(imdata);
    for k=1:length(xbox)
      for j=1:length(ybox)
%    for k=1:1
%      for j=1:1
       w=(k-1)*length(ybox)+j;


% If "redo" = 0, then I've already tried unsuccessfully to find
% correlation peaks for this box and there's no need trying again.
% So, only continue if redo(j,k)==1.

       if redo(j,k)==1

% Review the ang & vel data in the boxes surrounding (j,k)
% to see what the average values are.

           anglist=[];
           vellist=[];
           for g=(k-1):(k+1)
               for h=(j-1):(j+1)
                   if( g>0 & g<=length(xbox) & h>0 & h<=length(ybox)&(g~=k | h~=j))
                       anglist=[anglist;ang(h,g)];
                       vellist=[vellist;vel(h,g)];
```

```
                    end
                end
            end
            ff=[];
            fd=[];
            ff=find(anglist);
            fd=find(vellist);

% If all of the surrounding boxes are also zero'd (no vector
% yet) then skip this box and come back to it on the next pass
% in "zzz".

            if(ff~=[] & fd~=[])

% Find the average angle and velocity of the "non-zero" neighboring
% vectors.

                avgang=mean(anglist(ff));
                avgvel=mean(vellist(fd));

% If ang and vel at (j,k) are within a fraction of the
% distance to the current limits from the average values
% around it, then don't redo it...else...proceed

                nmaxang=avgang+(max_angle-avgang)*zzz;
                if min_angle>=0
                    nminang=avgang-(avgang-min_angle)*zzz;
                else
                    nminang=avgang-(avgang-0.0)*zzz;
                end
                nmaxvel=avgvel+(max_velocity-avgvel)*zzz;
                nminvel=avgvel-(avgvel-min_velocity)*zzz;

                if( ang(j,k)>nmaxang | ang(j,k)<nminang ...
                        | vel(j,k)>nmaxvel | vel(j,k)<nminvel )

% Now review the correlation analysis done in CCcorrbox3.m, this time
% searching within a tighter min/max angle and velocity band to
% see if a new peak can be found.  The new peak must be significant
% (within some threshold of the previous peak).

% Use the original min/max velocity & angle limits to compare any
% new peaks to the previous maximum.
% Compute delta yst= original yst - new yst for defining
% the reduced search area.  yst, yend, etc... are available from Ddata.

                    yst=Ddata(w,1);
                    yend=Ddata(w,2);
                    xst=Ddata(w,3);
                    xend=Ddata(w,4);

                    if nminvel>=0.0
                      dyst=abs(yst-round(lftynew(j,k)+nminvel*pulse_dt...
                        *cos(max([abs(90-nmaxang) abs(90-nminang)])...
                        *pi/180)/pixel_height-0.5*(box_height-1)));
                    else
                      dyst=abs(yst-round(lftynew(j,k)+...
                                nminvel*pulse_dt/pixel_height-...
                                0.5*(box_height-1)));
                    end

                    if (nminang<=90 & nmaxang>=90)
                        dyend=abs(yend-round(lftynew(j,k)...
                          +nmaxvel*pulse_dt/pixel_height ...
                          +0.5*(box_height-1)));
                        dxst=abs(xst-round(lftxnew(j,k)-nmaxvel*pulse_dt...
                          *sin((nmaxang-90)*pi/180)/pixel_width...
                          -0.5*(box_width-1)));
                        dxend=abs(xend-round(lftxnew(j,k)+nmaxvel*pulse_dt...
                          *sin((90-nminang)*pi/180)/pixel_width...
                          +0.5*(box_width-1)));

% If the limits don't include 90 degrees, need to adjust the
% region.

                    else
                        dyend=abs(yend-round(lftynew(j,k)+nmaxvel*pulse_dt...
                          *cos(min([abs(90-nmaxang) abs(90-nminang)])...
                          *pi/180)/pixel_height+0.5*(box_height-1)));
                        if(nminang>90)
                            dxst=abs(xst-round(lftxnew(j,k)-nmaxvel*pulse_dt...
                                *sin((nmaxang-90)*pi/180)/pixel_width...
                                -0.5*(box_width-1)));
                            dxend=abs(xend-round(lftxnew(j,k)-nminvel*pulse_dt...
                                *sin((nminang-90)*pi/180)/pixel_width...
                                +0.5*(box_width-1)));
                        else
                            dxst=abs(xst-round(lftxnew(j,k)+nminvel*pulse_dt...
                                *sin((90-nmaxang)*pi/180)/pixel_width...
                                -0.5*(box_width-1)));
                            dxend=abs(xend-round(lftxnew(j,k)+nmaxvel*pulse_dt...
                                *sin((90-nminang)*pi/180)/pixel_width...
                                +0.5*(box_width-1)));
                        end
                    end

% If this region extends beyond the limits of imdata, then can't
%  process...or...if the entire search region is only 3x3 or less,
% there's no sense in recorrelating it.  Ddata(w,5:6) contains
% the size of D.  If it's 0x0 then it's beyond bounds.

                    Dm=Ddata(w,5);
                    Dn=Ddata(w,6);
                    if ( Dm<4 & Dn<4 )
                        newdifx=0.0;
                        newdify=0.0;
                        newang=0.0;
                        newvel=0.0;
                        redo(j,k)=0;
                    else

% Retrieve the cross-correlation map "D" from the Ddata archive.

                        D=[];
                        D=reshape(Ddata(w,7:(6+Dm*Dn)),Dm,Dn);

% Retrieve the corrpeaks matrix from the Pksdata archive.

                        corrpeaks=[];
                        corrpeaks=reshape(Pksdata(w,1:(Dm*Dn)),Dm,Dn);

% Find the maximum value of D, only looking at peaks

                        summit=max(max(D.*corrpeaks));

% Create a smaller D matrix only covering the limited region
% being searched on this pass

                        Dpart=D((1+dyst):(Dm-dyend),(1+dxst):(Dn-dxend));

% Look in Dpart to find any peak
                        corrpeakspart=corrpeaks((1+dyst):(Dm-dyend)...
                                ,(1+dxst):(Dn-dxend));
                        nextpk=max(max(Dpart.*corrpeakspart));

% If there were no peaks, or if there is no new peak within "reduce" of
% "summit", then there's no reason to continue

                        if (find(corrpeaks)==[] | find(corrpeakspart)==[]...
                                 | summit*reduce>nextpk)
                            newdifx=0.0;
                            newdify=0.0;
                            newang=0.0;
                            newvel=0.0;
                            redo(j,k)=0;
                        else

% Find the position of this maximum in D, correcting Dpart
% indices for xst and yst

                            [a,b]=max(Dpart.*corrpeakspart);
                            [c,d]=max(a);
                            colmx=d+dxst;
                            rowmx=b(d)+dyst;

% Better this with a sub-pixel correlation using an area centroid
% fit to the near-maximum portion of the D matrix.
% First find the extent of the maximum portion of the
% correlation.

                            Drat=[];
                            Drat=D/summit;
                            dir=[0 1;-1 1;-1 0;-1 -1;0 -1;1 -1;1 0;1 1];
                            steps=[];
                            val=[];
                            for m=1:8
                              [steps(m),val(m)]=descend(Drat,colmx,rowmx,dir(m,:));
                            end

% Proceed to form a region about this max point using corrmap.

                            ind1=[8 1 2 ; 2 3 4 ; 4 5 6 ; 6 7 8];
                            for l=1:4
                               maxext3(l)=max([steps(ind1(l,1))...
                                       steps(ind1(l,2)) steps(ind1(l,3))]);
                            end
                            Dfit=[];
                            [Dfit]=corrmap(Drat,colmx,rowmx,steps,mean(val),maxext3);

% Adjust the max position (colmx,rowmx) in the case that size(Dfit)<
% size (Drat) to follow the true max position.

                            ymaxdfit=maxext3(2)+1;
                            xmaxdfit=maxext3(3)+1;

% Further refine the correlation map by zeroing out all pixels
% with a value < mean(val).  Then perform a centroid fit.

                            Dfit=Dfit-mean(val);
                            Dfit(find(Dfit<0))=zeros(size(find(Dfit<0)));
                            ff=[];
                            ff=find(Dfit<(0.75*max(max(Dfit))));
                            Dfit(ff)=zeros(size(ff));
                            [xcntg,ycntg]=centroidfit(Dfit);

% Store the resulting correlation refinement in newdifx,newdify

                           newdifx=xst+colmx-1+xcntg-xmaxdfit+...
                               0.5*(box_width-1)-lftxnew(j,k);
                           newdify=yst+rowmx-1+ycntg-ymaxdfit+...
                               0.5*(box_height-1)-lftynew(j,k);
                           compl=newdifx*pixel_width+newdify*pixel_height*i;
                           newang=angle(compl)*180/pi;
                           newvel=abs(compl)/pulse_dt;

% If this newdifx,newdify results in a vel or ang outside of max/min
% then delete it.

                           if(newvel>nmaxvel | newvel<nminvel |...
                               newang>nmaxang | newang<nminang)
                               newdifx=0.0;
                               newdify=0.0;
                               newang=0.0;
                               newvel=0.0;
                           end
                       end
                   end

% If a new vel/ang are found, take it

                   if (newvel~=0 | newang~=0)
                       if (vel(j,k)~=0 & ang(j,k)~=0)
                           numnew=numnew+1;
                       end

                       difx(j,k)=newdifx;
                       dify(j,k)=newdify;
                       ang(j,k)=newang;
                       vel(j,k)=newvel;
                       goodbox(j,k)=1;
                       changed(j,k)=1;
                   end
               end    % within reduced vel/ang range conditional
           end    % no neighboring vector conditional
       end    % end of redo conditional
     end    % end of j loop
   end    % end of k loop

% Determine the total number of boxes that are still zero and
% the % of boxes that are filled with averages rather
% than data.

   numz=length(find(ang==0 & vel==0));
```

```
fprintf('\n%.2f\t%d\t\t%d\t%d\t%.0f',zzz,numnew,numz,length(xbox)*length(ybox),(100*numz/length(xbox
)/length(ybox)));
    numnewtot=numnewtot+numnew;

  end    % end of "zzz" FOR loop

  fprintf('\nTotal changes = %d',numnewtot);
  fprintf('\nUnique changes = %d\n',length(find(changed)));

% Show on LTSWin plot the boxes that have data and those
% that have "zeros".  The new vectors are green.

    if LTSWin==0
        fprintf('No LTSWin available for displaying boxes \n');
    else
        figure(LTSWin);
        mbox=length(ybox);
        nbox=length(xbox);
        hold on
        xzero=[];
        yzero=[];
        for k=1:length(xbox)
            for j=1:length(ybox)
                if goodbox(j,k)==0
                    xzero=[xzero;(xbox(k)-ulxrect+1)];
                    yzero=[yzero;(ybox(j)-ulyrect+1)];
                end
            end
        end
        jpbquiv((xbox-ulxrect+1),(ybox-ulyrect+1),difx,dify,scale,'-g');
        if xzero~=[]
            plot(xzero,yzero,'ro');
        end
        hold off
    end

% Open a HistWin for plotting velocity & angle histograms

    if (HistWin==0)
        HistWin=figure;
        orient landscape
    else
        figure(HistWin);
        clf
    end
    set(HistWin,'name',filename);

% Plot angle histogram

    axes('units','normalized','position',[.05 .1 .9 .3])
    hist(ang(find(goodbox)),30);
    set(gca, 'xtick',[-180:10:180]);
    grid on
    set(gca, 'xlim',[(min(min(ang(find(goodbox))))-5) (max(max(ang(find(goodbox))))+5)]);
    mnangle=mean(mean(ang(find(ang))));
    mang=num2str(mnangle);
    stdangle=sqrt(mean(mean((ang(find(ang))-mnangle).*(ang(find(ang))-mnangle))));
    stdang=num2str(stdangle);
    nmin=num2str(min_angle);
    nmax=num2str(max_angle);
    title(['mean=',mang,'std=',stdang,' min=',nmin,' max=',nmax]);
    xlabel('Angles in degrees');

% Plot velocity histogram

    axes('units','normalized','position',[.05 .55 .9 .3])
    hist(vel(find(goodbox)),30);
    set(gca, 'xtick',[-5:.5:10]);
    grid on
    set(gca, 'xlim',[min_velocity max_velocity]);
    mnvel=mean(mean(vel(find(vel))));
    mvel=num2str(mnvel);
    stdveloc=sqrt(mean(mean((vel(find(vel))-mnvel).*(vel(find(vel))-mnvel))));
    stdvel=num2str(stdveloc);
    nmin=num2str(min_velocity);
    nmax=num2str(max_velocity);
    title(['mean=',mvel,'std=',stdvel,' min=',nmin,' max=',nmax]);
    xlabel('Velocity in m/s');

end    % end of inputs conditional
```

```
%
%       Title: CCredovect2.m
%       Author:  Jeffrey Bons
%       Parent program: jpbpiv3.m
%       Child subroutine(s): centroidfit.m, descend.m, & corrmap.m
%       Date: 25 Mar 1997
%
% This program is used to redo a single vector processed
% by CCcorrbox3.m & CCrecorrbox4.

% Must have vel & ang data & box coordinates & parameters to proceed.

if (xbox==[] | pixel_width==[] | pixel_height==[] | box_width==[] |...
    box_height==[] | pulse_dt==[] | max_velocity==[] |...
    min_velocity==[] | min_angle==[] | max_angle==[] |...
    rotfreq_rps==[] | xcntr_pix==[] | ycntr_pix==[] | absmean==[] | ...
    ang==[] | vel==[] | lftxnew==[] | lftynew==[] | reduce==[] |...
    goodbox==[] | ulyrect==[] | ulxrect==[] | lryrect==[] | lryrect==[])

    fprintf('Error: Need velocities, boxes and parameters first!!\n');
else

% User can specify the vector to redo by mouse-clicking the vector in
% the LTSWin figure.

  if LTSWin==0
    fprintf('No LTSWin available for displaying boxes \n');
  else
    figure(LTSWin);

    fprintf('Mouse-select 1 vector to redo (exit=below boundary): \n');
    [xredo,yredo]=ginputc(1);
    xboxredo=ceil(xredo/box_width);
    yboxredo=ceil(yredo/box_height);

    numchng=0;
    numlook=0;
    while xboxredo>=0 & xboxredo<=length(xbox) & ...
                yboxredo>=0 & yboxredo<=length(ybox)

    [mmax,nmax]=size(imdata);
    k=xboxredo;
    j=yboxredo;
    w=(k-1)*length(ybox)+j;

% Review the ang & vel data in the boxes surrounding (j,k)
% to see what the average values are.

    anglist=[];
    vellist=[];
    for g=(k-1):(k+1)
        for h=(j-1):(j+1)
            if( g>0 & g<=length(xbox) & h>0 & h<=length(ybox)&(g~=k | h~=j))
                anglist=[anglist;ang(h,g)];
                vellist=[vellist;vel(h,g)];
            end
        end
    end
    ff=[];
    fd=[];
    ff=find(anglist);
    fd=find(vellist);

    avgang=mean(anglist(ff));
    avgvel=mean(vellist(fd));

    fprintf('\nAverage of neighbors: vel = %.2f  ang = %.1f\n',avgvel,avgang);
    fprintf('Current vector: vel = %.2f  ang = %.2f\n',vel(j,k),ang(j,k));
    prevvel=vel(j,k);
    prevang=ang(j,k);

% Read in the D matrix from the Ddata archive

    yst=Ddata(w,1);
    yend=Ddata(w,2);
    xst=Ddata(w,3);
    xend=Ddata(w,4);
    Dm=Ddata(w,5);
    Dn=Ddata(w,6);


% If this region extends beyond the limits of imdata, then can't
%  process...

    if (xend>nmax | xst<1 | yst<1 | yend>mmax)
        fprintf('Correlating region beyond bounds...\n');
    else

% Retrieve the cross-correlation map "D" from the Ddata archive.

        D=[];
        D=reshape(Ddata(w,7:(6+Dm*Dn)),Dm,Dn);

% Retrieve the corrpeaks matrix from the Pksdata archive.

        corrpeaks=[];
        corrpeaks=reshape(Pksdata(w,1:(Dm*Dn)),Dm,Dn);

% Print all the peaks and their corresponding ang vel for selection

        DC=[];
        DC=D.*corrpeaks;
        fprintf('#\tintens/summit\testvel\testang\n');
        for v=1:length(find(corrpeaks))
            [a,b]=max(DC);
            [intpk(v),colpk(v)]=max(a);
            rowpk(v)=b(colpk(v));
            estdifx=xst+colpk(v)-1+0.5*(box_width-1)-lftxnew(j,k);
            estdify=yst+rowpk(v)-1+0.5*(box_height-1)-lftynew(j,k);
            compl=estdifx*pixel_width+estdify*pixel_height*i;
            estang=angle(compl)*180/pi;
            estvel=abs(compl)/pulse_dt;
            DC(rowpk(v),colpk(v))=0;
            if 100*intpk(v)/intpk(1)>40
                fprintf('%d\t%.0f\t\t%.1f\t%.1f\n',v,...
                       100*intpk(v)/intpk(1),estvel,estang);
            end
        end

% Show the D matrix in the window CorrWin

        if (CorrWin==0)
            CorrWin=figure;
        else
            figure(CorrWin)
            clf
        end

        axes('position',[.1 .1 .8 .4]);
        contour(D,20);
        axes('position',[.1 .55 .8 .4]);
        surf(D);
        view(38,60);

% Allow user to select vector of choice

        numsel=[];
        numsel=input('Select vector by # <0 = none & delete current>: ');
        if isempty(numsel) | numsel==0 | numsel>length(find(corrpeaks))
            numsel=0;
            difx(j,k)=0.0;
            dify(j,k)=0.0;
            ang(j,k)=0.0;
            vel(j,k)=0.0;
            goodbox(j,k)=0;
        else

% If none were selected, delete the current (if any) vector.  If a new
% vector is selected...refine the ang and velocity estimates.

% First find the extent of the maximum portion of the
% correlation.

            Drat=[];
            Drat=D/intpk(1);
            dir=[0 1;-1 1;-1 0;-1 -1;0 -1;1 -1;1 0;1 1];
            steps=[];
            val=[];
            for m=1:8
                [steps(m),val(m)]=descend(Drat,colpk(numsel),rowpk(numsel),dir(m,:));
            end

% Proceed to form a region about this max point using corrmap.

            indl=[8 1 2 ; 2 3 4 ; 4 5 6 ; 6 7 8];
            for l=1:4
                maxext3(l)=max([steps(indl(l,1))...
                         steps(indl(l,2)) steps(indl(l,3))]);
            end
            Dfit=[];
            [Dfit]=corrmap(Drat,colpk(numsel),rowpk(numsel),steps,mean(val),maxext3);

% Adjust the max position (colpk(numsel),rowpk(numsel)) in the case
% that size(Dfit)< size (Drat) to follow the true max position.

            ymaxdfit=maxext3(2)+1;
            xmaxdfit=maxext3(3)+1;

% Further refine the correlation map by zeroing out all pixels
% with a value < mean(val).  Then perform a centroid fit.

            Dfit=Dfit-mean(val);
            Dfit(find(Dfit<0))=zeros(size(find(Dfit<0)));
            ff=[];
            ff=find(Dfit<(0.75*max(max(Dfit))));
            Dfit(ff)=zeros(size(ff));
            [xcntg,ycntg]=centroidfit(Dfit);

% Store the resulting correlation refinement in difx,dify
% for processing by plotrotvel.m

            difx(j,k)=xst+colpk(numsel)-1+xcntg-xmaxdfit+...
                        0.5*(box_width-1)-lftxnew(j,k);
            dify(j,k)=yst+rowpk(numsel)-1+ycntg-ymaxdfit+...
                        0.5*(box_height-1)-lftynew(j,k);
            compl=difx(j,k)*pixel_width+dify(j,k)*pixel_height*i;
            ang(j,k)=angle(compl)*180/pi;
            vel(j,k)=abs(compl)/pulse_dt;
            goodbox(j,k)=1;
            fprintf('New vector: vel = %.2f ang = %.2f\n',vel(j,k),ang(j,k));
        end
    end

    fprintf('Mouse-select 1 vector to redo (exit=below boundary): \n');
    figure(LTSWin);
    [xredo,yredo]=ginputc(1);
    xboxredo=ceil(xredo/box_width);
    yboxredo=ceil(yredo/box_height);

    if abs(prevvel-vel(j,k))>0.1 | abs(prevang-ang(j,k))>0.5
        numchng=numchng+1;
    end
    numlook=numlook+1;
    end  % end of while loop

    fprintf('\nBoxes checked: %d\n',numlook);
    fprintf('Boxes changed: %d\n',numchng);

% Show on LTSWin plot the new vector or zero with the old.

    figure(LTSWin);
    hold on
    xzero=[];
    yzero=[];
    for k=1:length(xbox)
        for j=1:length(ybox)
            if goodbox(j,k)==0
                    xzero=[xzero;(xbox(k)-ulxrect+1)];
                    yzero=[yzero;(ybox(j)-ulyrect+1)];
            end
        end
    end
    jpbquiv((xbox-ulxrect+1),(ybox-ulyrect+1),difx,dify,scale,'-y');
    if xzero~=[]
            plot(xzero,yzero,'ro');
    end
    hold off

% Open a HistWin for plotting velocity & angle histograms
```

```
  if (HistWin==0)
    HistWin=figure;
    orient landscape
  else
    figure(HistWin);
    clf
  end
  set(HistWin,'name',filename);

% Plot angle histogram

  axes('units','normalized','position',[.05 .1 .9 .3])
  hist(ang(find(goodbox)),30);
  set(gca, 'xtick',[-180:10:180]);
  grid on
  set(gca, 'xlim',[(min(min(ang(find(goodbox))))-5) (max(max(ang(find(goodbox))))+5)]);
  mnangle=mean(mean(ang(find(ang))));
  mang=num2str(mnangle);
  stdangle=sqrt(mean(mean((ang(find(ang))-mnangle).*(ang(find(ang))-mnangle))));
  stdang=num2str(stdangle);
  nmin=num2str(min_angle);
  nmax=num2str(max_angle);
  title(['mean=',mang,'std=',stdang,' min=',nmin,' max=',nmax]);
  xlabel('Angles in degrees');

% Plot velocity histogram

  axes('units','normalized','position',[.05 .55 .9 .3])
  hist(vel(find(vel)),30);
  set(gca, 'xtick',[-5:.5:10]);
  grid on
  set(gca, 'xlim',[min_velocity max_velocity]);
  mnvel=mean(mean(vel(find(vel))));
  mvel=num2str(mnvel);
  stdveloc=sqrt(mean(mean((vel(find(vel))-mnvel).*(vel(find(vel))-mnvel))));
  stdvel=num2str(stdveloc);
  nmin=num2str(min_velocity);
  nmax=num2str(max_velocity);
  title(['mean=',mvel,'std=',stdvel,' min=',nmin,' max=',nmax]);
  xlabel('Velocity in m/s');
 end
end



%
%       Title:  CCprepplot.m
%       Author:  Jeffrey Bons
%       Parent program: jpbpiv3
%       Child subroutine(s):
%       Date: 6 Mar 1997
%
% This program prepares data to be plotted as vectors,
% contour plots, and histograms using CCplot.m
% given the information yeilded from CCcorrbox.m & CCcleanup.m

% If these programs haven't been run yet, can't plot...

if (xbox==[] | pixel_width==[] | pixel_height==[] |...
  xcntr_pix==[] | ycntr_pix==[] | pulse_dt==[] |...
  ang==[] | vel==[] | lftxnew==[] | lftynew==[] |...
  difx==[] | dify==[] | cammotion==[])
  fprintf('Error: Need velocities, boxes and parameters first!!\n');
else

% Now, I want to compute the coordinates of each box center in a frame
% of reference fixed to the rotating passage rather than the
% x,y relative to the corner of the camera frame.
% To do this, I must enter a point which designates the center of
% the right test section view then I can compute the angle "alpha"
% between the two reference frames (the test section & the camera
% frame).

% For stationary PIV, if "alpha" already exists, then use it.

  if rotfreq_rps==0 & alpha~=[]
    fprintf('Using current alpha=%.2f degrees\n',alpha*180/pi);
  else
```

```
% For rotating PIV, allow use of current alpha if desired.

       ansq='n';
       if alpha~=[]
         fprintf('Current alpha is %.2f degrees\n',alpha*180/pi);
         ansq=input('Use current alpha? (y/n) <n>: ','s');
         if isempty(ansq)
             ansq='n';
         end
       end

% If ImageWin is available use ginputc, if not then enter the
% pixel coordinates manually.

       if ansq=='n'
        if ImageWin==0
            xts=input('Enter x pixel of right test section image center: ');
            yts=240;
        else
            figure(ImageWin);
            fprintf('Select right test section image center...\n');
            [xts,yts]=ginputc(1);
            yts=240;
        end

% Now compute the angle between the radial (axial) direction in the
% test section frame and the x direction in the camera frame.

% If the portion of the test section in view is NOT the center
% then the calculation is more convoluted...

%         TSoffset=input('Enter TS shield offset in um (+=TS, -=LS) <0>');

%         if isempty(TSoffset)
%            TSoffset=0;
%       end

        TSoffset=0;
        if TSoffset==0
            alpha=atan((yts-ycntr_pix)*pixel_height/(xts-xcntr_pix)/pixel_width);
        else
            alpha=atan((yts-ycntr_pix)*pixel_height/(xts-xcntr_pix)/pixel_width);
        end

% alpha is in radians

        if(alpha<0)
            alpha=pi+alpha;
        end
        fprintf('Test section angle is, alpha = %.2f\n',alpha*180/pi);
      else
        fprintf('Using current alpha...\n');
      end
    end

% Now with the angle alpha determined, I can convert from x,y in
% frame of reference to coordinates in test section frame.
% The distance from the test section inlet to the alignment tool
% cross is 38200 microns in the radial direction (notebook pg. 63).
% This is considered to be the "home" position of the camera.
% If the picture being processed is offset from
% the "home" position of the camera, the amount of offset
% is =cammotion from the front panel (assume it is all radial).

    delts=38200+cammotion;

% Compute the coordinates of the upper-left corner of the camera
% frame (x=0,y=0) in the test section coordinates (del,eps)

    dts=sqrt((xts-1)^2*pixel_width^2+(yts-1)^2*pixel_height^2);
    epscrnr=dts*cos((pi/2-alpha)+...
                atan((yts-1)*pixel_height/(xts-1)/pixel_width));
    delcrnr=delts-dts*sin((pi/2-alpha)+...
                atan((yts-1)*pixel_height/(xts-1)/pixel_width));

% Now loop through all the box's rotated center coordinates
% (lftxnew,lftynew) and convert them to test section
% coordinates by first referencing them to the camera
% frame corner and then to the new coord system.
% Also, decompose the pixel distances (difx,dify) into
% distances relative to the test section axis using alpha.
```

```
    delbox=[];
    epsbox=[];
    difdel=[];
    difeps=[];
    for k=1:length(xbox)
        for j=1:length(ybox)
            difdel(j,k)=sin(alpha)*dify(j,k)*pixel_height+...
                       cos(alpha)*difx(j,k)*pixel_width;
            difeps(j,k)=cos(alpha)*dify(j,k)*pixel_height-...
                       sin(alpha)*difx(j,k)*pixel_width;
            betabx=atan(lftxnew(j,k)*pixel_width/lftynew(j,k)...
                       /pixel_height)-(pi/2-alpha);
            dp=sqrt((lftxnew(j,k)-1)^2*pixel_width^2+...
                       (lftynew(j,k)-1)^2*pixel_height^2);
            epsbox(j,k)=epscrnr-dp*sin(betabx);
            delbox(j,k)=delcrnr+dp*cos(betabx);
        end
    end

% First compute the velocity and angle values in the test section
% frame of reference.
% Compute angle of vector

    compl=[];
    TSang=[];
    TSvel=[];
    compl=difeps+difdel*i;
    TSang=-angle(compl)*180/pi+90;

% Divide distance in microns by deltat in microseconds to get m/s

    TSvel=abs(compl)/pulse_dt;
end




%
%       Title:  CCplot3.m
%       Author:  Jeffrey Bons
%       Parent program: jpbpiv3
%       Child subroutine(s):
%       Date: 25 Mar 1997
%
% This program plots vectors and contour plots and histograms
% given the information from CCprepplot.m

% If these programs haven't been run yet, can't plot...

if (epsbox==[] | delbox==[] | difeps==[] | difdel==[] |...
    max_velocity==[] | min_velocity==[] | min_angle==[] | max_angle==[] |...
    TSang==[] | TSvel==[] | goodbox==[] | alpha==[])
    fprintf('Error: Need velocities, boxes and parameters first!!\n');
else

% Plot the data, nondimensionalized by the
% tube hydraulic diameter = 10000 um.

% Open a PlotWin for plotting velocity vectors

    if (PlotWin==0)
        PlotWin=figure;
        orient landscape
    else
        figure(PlotWin);
        clf
    end
    set(PlotWin,'name',filename);

% Plot vectors

    axes('units','normalized','position',[.1 .1 .4 .63]);
    set(gca,'Box','on');
    hold on
    axis('equal');
    set(gca, 'xtick',[-.5:.05:.5]);
    set(gca, 'ytick',[0:0.05:10]);
    grid on
    jpbquiv(epsbox/10000,delbox/10000,difeps/10000,difdel/10000,scale,'-y');
    xlabel('y/d from centerline');
    ylabel('x/d from passage inlet');
    title(['Velocity Vectors for ',filename,' (o=no vector)']);
    text('units','inches','position',[-.5,0],'rotation',(90),'string','Leading');
    text('units','inches','position',[4.5,.6],'rotation',(-90),'string','Trailing');

% Identify the boxes without vectors

    hold on
    [mplot,nplot]=size(TSang);
    epszero=[];
    delzero=[];
    for k=1:nplot
        for j=1:mplot
            if goodbox(j,k)==0
                epszero=[epszero;epsbox(j,k)/10000];
                delzero=[delzero;delbox(j,k)/10000];
            end
        end
    end
    if epszero~=[]
        plot(epszero,delzero,'ro');
    end
    hold off

% Enter the run parameters

    if Renum==[]
        Renum=input('Enter Reynolds number: ');
        Rotnum=input('Enter Rotation number: ');
        zoverdh=input('Enter z/dh: ');
    end
    relab=['Reynolds # = ',num2str(Renum)];
    rotlab=['Rotation # = ',num2str(Rotnum)];
    zdlab=['z/dh = ',num2str(zoverdh)];
    text('units','normalized','position',[.5,1.3],'string',[relab]);
    text('units','normalized','position',[.5,1.26],'string',[rotlab]);
    text('units','normalized','position',[.5,1.22],'string',[zdlab]);
    text('units','normalized','position',[-.1,1.3],'string',['Date: ',date]);
    text('units','normalized','position',[-.1,1.26],'string',['File: ',filename]);
    perc=num2str(100*length(find(goodbox))/(mplot*nplot));
    perclab=[perc ,'% boxes with vectors'];
    text('units','normalized','position',[-.1,1.22],'string',[perclab]);
    nmin=num2str(min_angle-alpha*180/pi);
    nmax=num2str(max_angle-alpha*180/pi);
    anglab=['Angle search range: ',nmin,' < angle < ',nmax];
    text('units','normalized','position',[-.1,1.16],'string',[anglab]);
    nmin=num2str(min_velocity);
    nmax=num2str(max_velocity);
    vellab=['Velocity search range: ',nmin,' < velocity < ',nmax];
    text('units','normalized','position',[-.1,1.12],'string',[vellab]);

% Calculate axial and transverse velocity components in m/s

    TSaxvel=TSvel.*cos(TSang*pi/180);
    TStrvel=TSvel.*sin(TSang*pi/180);

% Plot mean axial velocity as a function of y/d

    axes('units','normalized','position',[.575 .1 .375 .175]);
    set(gca,'Box','on');
    mneps=[];
    mnaxv=[];
    mntrv=[];
    for k=1:length(xbox)
        mneps(k)=mean(epsbox(find(goodbox(:,k)),k)/10000);
        mnaxv(k)=mean(TSaxvel(find(goodbox(:,k)),k));
        mntrv(k)=mean(TStrvel(find(goodbox(:,k)),k));
    end
    plot(mneps,mnaxv);
    grid on
    set(gca, 'xtick',[-.5:.1:.5]);
    set(gca, 'ytick',[-5:.5:10]);
    xlabel('y/d from centerline');
    axmean=num2str(mean(mean(TSaxvel(find(goodbox)))));
    title(['Mean Axial Velocity vs. y/d  (MEAN = ',axmean,' m/s)']);

% Plot mean transverse velocity as a function of y/d

    axes('units','normalized','position',[.575 .325 .375 .175]);
    set(gca,'Box','on');
    plot(mneps,mntrv);
    grid on
    set(gca, 'xtick',[-.5:.1:.5]);
    set(gca, 'ytick',[-10:.2:10]);
    set(gca, 'xticklabels',['']);
    trmean=num2str(mean(mean(TStrvel(find(goodbox)))));
    title(['Mean Transverse Velocity vs. y/d  (MEAN = ',trmean,' m/s)']);

% Plot angle histogram

    axes('units','normalized','position',[.575 .8 .375 .12])
    hist(TSang(find(goodbox)),[-90:2.5:270]);
    set(gca, 'xtick',[-90:10:270]);
    grid on
    mnang=mean(mean(TSang(find(goodbox))));
    mang=num2str(mnang);
    stdangl=sqrt(mean(mean((TSang(find(goodbox))-mnang).*(TSang(find(goodbox))-mnang))));
    stdang=num2str(stdangl);
    sect2=['Angle Histogram (deg): mean='];
    nmin=num2str(min_angle-alpha*180/pi);
    nmax=num2str(max_angle-alpha*180/pi);
    title([sect2,mang,' ,std dev=',stdang]);
    set(gca, 'xlim',[(min(min(TSang(find(goodbox))))-5) (max(max(TSang(find(goodbox))))+5)]);
%    set(gca, 'xlim',[(min_angle-alpha*180/pi) (max_angle-alpha*180/pi)]);

% Plot velocity histogram

    axes('units','normalized','position',[.575 .6 .375 .12])
    hist(TSvel(find(goodbox)),[0:.2:10]);
    set(gca, 'xtick',[0:.5:10]);
    grid on
    mnvel=mean(mean(TSvel(find(goodbox))));
    mvel=num2str(mnvel);
    stdvelo=sqrt(mean(mean((TSvel(find(goodbox))-mnvel).*(TSvel(find(goodbox))-mnvel))));
    stdvel=num2str(stdvelo);
    sect3=['Velocity Histogram (m/s): mean='];
    nmin=num2str(min_velocity);
    nmax=num2str(max_velocity);
    title([sect3,mvel,' ,std dev=',stdvel]);
    set(gca, 'xlim',[min_velocity max_velocity]);

end



%
%       Title: crclcntr.m
%       Author: Jeffrey Bons
%       Parent program: jpbpiv1.m
%       Child subroutine(s):
%       Date: 22 July 1996
%
% This program solves for the center of the circle of a given
% radius, knowing 2 points on the circle.  Also need to know
% which "side" (+1=down or -1=up) of the points to look for
% the center.

% Need certain inputs from jpbpiv1 parameter list to process

if(lftcrossx==[] | lftcrossy==[] | corrxy==[] | hfact==[] |...
   vfact==[] | rad_cross==[])
    fprintf('ERROR: Set rad_cross or run corrimagjb or xyresol..\n');
else
    rad1=rad_cross;

% Read in the (x,y) pixel coordinates of left wire cross and
% convert to micrometers

    x1a=lftcrossx*hfact;
    y1a=lftcrossy*vfact;

% Calculate location of right cross from corrimagjb deltax,y data

    x1b=x1a+corrxy(1,1)*hfact;
    y1b=y1a+corrxy(1,2)*vfact;

% Compute the center coordinates by first guessing yc,
% then solving for xc via the circle equation for (x1b,y1b).
% Then use the (x1a,y1a) circle equation to "re"-solve for
% yc...compare the 2 and iterate until they converge...
% Since 2 centers can be found given only 2 points and a radius
% start with a yc1 guess which will converge to the "correct"
% center for the figure (assuming a center of rotation which
% is above the image [negative y]).

    side=-1;
    yc1=y1a+side*400000;
    del=1;
    it=1;
    xc=1;
    yc2=1;
    while (del>1e-9 & it<120 & imag(xc)==0 & imag(yc2)==0)
        it=it+1;
        xc=x1b-sqrt(rad1^2-(y1b-yc1)^2); %take the negative root
        yc2=y1a-sqrt(rad1^2-(x1a-xc)^2);
        del=abs((yc1-yc2)/yc2);
        if(rem(it,2)==0)
            fprintf('it=%d xc=%.4f yc=%.4f del=%.4f\n',it,xc/hfact,yc2/vfact,del);
        end
        yc1=yc2;
    end
    fprintf('\nTotal # of iterations= %d\n',it);

% With this value for yc, solve for xc

    yc=yc1;
    xc=x1b-sqrt(rad1^2-(y1b-yc)^2); %again take the negative root

% print out xc,yc result in pixels

    rotcentx=xc/hfact;
    rotcenty=yc/vfact;
    fprintf('Center is at:   x = %.4f pixels\n',rotcentx);
    fprintf('\t\ty = %.4f pixels\n',rotcenty);
    clear rad1 xc yc yc1 yc2 x1a y1a x1b y1b
end
```